SNANA User's Manual: Simulation, Lightcurve Fitter & Cosmology Fitter

R. Kessler

December 27, 2010

Contents

1 Introduction

This manual describes how to use the lightcurve simulation and fitter in the SNANA product. This code was originally developed for the SDSS-II Supernova Survey, and then it was modified to simulate and fit SN Ia lightcurves for non-SDSS surveys. Current SN models include MLCS2k2 [1], SALT-II [2], Δm_{15} [3], stretch [4], and two-stretch[5].

The simulation is designed to be fast, generating a few dozen lightcurves per second, and still provide an accurate and realistic description of supernova lightcurves. In particular, the simulation accounts for variations in noise, atmospheric transmission, and cadence. The reliability of the simulation is based on the accuracy of the "seeing conditions" that describes the seeing, sky-noise, zeropoints, and cadence. The conditions file is easy to prepare post-survey; predicting the conditions file before the survey is crucial to making reliable predictions for the lightcurve quality. Note that the simulation does not use pixels or images, although it makes use of information from the images.

2 SNANA Basics

From any DES or SDSS server at Fermilab, invoke the SNANA product with the command:

```
> setup snana
```

This command defines a few useful global environment variables:

```
> echo $SNANA_DIR/
/sdss/ups/prd/snana/v8_04/Linux

> echo $SNDATA_ROOT/
/data/dp47.b/data/sn/data/
```

At non-FNAL sites, you will have to define $SNANA_DIR and $SNDATA_ROOT as part of the installation. The environment variable $SNANA_DIR points to the software that is accessible to everyone, but write-protected. The SNANA developers use a cvs repository to share code, and an updated version is "cut" (i.e, released) on occasion to provide a stable software version for collaborators. SNANA is re-released as often as necessary (§ 9), as bugs are fixed and improvements are made. The current software version is v8_04 as indicated by $SNANA_DIR. You will likely have a future software version when you read this manual. If there is a problem with the current software version, you can fall back to an earlier version with

```
> setup snana v3_03
```

The main source codes for the simulation and fitter are

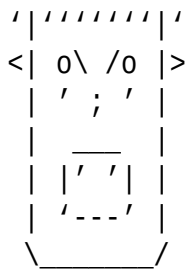
```
> $SNANA_DIR/src/snlc_sim.c
> $SNANA_DIR/src/snlc_fit.car
```

The environment variable $SNDATA_ROOT points to the user area. Everyone has write privilege here, so please be careful. The contents are explained in $SNDATA_ROOT/AAA_README.

2.1 Dr. Evil-ABORT-Face

The SNANA simulation and fitter programs have intensive error checking throughout the execution. If anything looks fishy, the program will abort with a message looking like

```
 FATAL[get_user_input]: Cannot open input file :
                  'sim_BLABLA.input'
```



```
ABORT program on Fatal Error.
```

While some of these aborts may at first seem frustrating, they are crucial for catching bugs as early as possible so that you don't waste months (years) doing something silly.

3 The SNANA Simulation: snlc_sim

3.1 Overview

For a rest-frame model of supernova, such as MLCS2k2, here is a brief overview of what the simulation does:

1. pick random luminosity parameter (Δ) and random extinction (A_V) according to measured distributions.
2. generate rest-frame light curve: U, B, V, R, I mag vs. time.
3. apply host-galaxy extinction to *UBVRI* mags.
4. add K-correction to transform *UBVRI* to observer-frame filters.
5. apply Galactic (MilkyWay) extinction.
6. apply zero-points to account for atmospheric transmission.
7. use PSF to compute sky noise and host-galaxy noise.

For an observer-frame mode such as SALT-II, steps 2-5 are combined into one generator.

3.2 Getting Started Quickly

In this section, you should be able to start simulating lightcurves in a few minutes. There are many options that may take some practice to use properly. The first step is to copy a sample input file to your private area,

```
> cp $SNDATA_ROOT/analysis/sample_input_files/MLCS/sim_[SURVEY].input.
```

where [SURVEY] is one of the surveys for which a sample sim-input file is available. Edit the file and change the GENVERSION name:

```
GENVERSION: CHANGE_ME
```

We recommend just adding your initials and/or project name so that you do not over-write somebody else's files. Now you can run the simulation, for example, with

```
> snlc_sim.exe  sim_SDSS.input
```

which should generated ten lightcurves using the MLCS2k2 model. The next step is to modify the input file to suit your needs. The input parameters are internally commented within the source code; for example, to get more information about the GENMODEL_ERRSCALE: keyword,

```
> grep GENMODEL_ERRSCALE $SNANA_DIR/src/snlc_sim.h
> grep GENMODEL_ERRSCALE $SNANA_DIR/src/snlc_sim.c
```

will help you trace the meaning of this variable. Please report variables that are not commented, or that have confusing comments. Don't hesitate contacting other people familiar with snlc_sim. Some of the light curve parameters are obscure (i.e, describing the distribution of extinction & Δ), and our best estimates of these parameters can change. To reduce the burden of tracking all of these parameters, defaults for these obscure parameters are stored in the file

```
$SNDATA_ROOT/models/snlc_sim.defaults  .
```

If you simply ignore these obscure parameters in your file, the "defaults" defined above will be used in the simulation. You can modify any parameter by defining the parameter explicitly in your input file.

The simulated lightcurves are located in $SNDATA_ROOT/SIM. Do NOT try 'ls' !!! Instead, try 'ls -d */' to see all versions. To avoid sifting through hundreds of versions from multiple users, I always use 'RK' as a prefix for my versions, and therefore I can see my versions with 'ls -d RK*/.' Each simulated version is located in a sub-directory named by your version. Recall that you have full write-privilege, so use caution. Each version has an auto-generated README file. If your version is called 'MYFIRSTSIM', then do

```
> more $SNDATA_ROOT/SIM/MYFIRSTSIM/MYFIRSTSIM.README
```

which contains a list of all your simulation options. Each simulated lightcurves is in a separate file; you can get a list of files with the commands

```
cd  $SNDATA_ROOT/SIM/MYFIRSTSIM/
ls  MYFIRSTSIM_SN*
   or
more MYFIRSTSIM.LIST
```

These files are rather verbose. The SNANA fitter reads them, but you may want to change to a less verbose format for external/private software. Translators have been written to convert to formats used by the original IDL-MLCS2k2 and by SALT-II.

3.3 Simulating Type Ia Supernova

The simulation of Type Ia supernova involves the selection of one of the following models:

```
GENMODEL:  mlcs2k2   # JRK 2007
GENMODEL:  SALT2     # Guy 2007
GENMODEL:  snoopy    # Burns 2010
GENMODEL:  stretch   # stretch a template
GENMODEL:  stretch2  # stretch pre- and post-max
KCOR_FILE:  <kcor filename>
```

The lower-case models indicate that rest-frame magnitudes are generated and K-corrections are used to transform into the observer-frame. Upper case (SALT2) indicates that observer-frame magnitudes are generated directly from the model without the need for K-corrections. The KCOR_FILE must reside

in $SNDATA_ROOT/kcor, and must be specified for all models, even observer-frame models such as SALT2. Although SALT2 does not use K-corrections, it uses the filter transmission curves and primary reference spectrum stored in the KCOR_FILE, and thus ensures that the same information is used for all models that point to the same KCOR_FILE.

For each Ia model, there are parameters to control the distributions of the SN brightness and color. Default parameters are given by GENMEAN_xxx, GENRANGE_xxx and GENSIGMA_xxx in the default sim-input file:

```
$SNDATA_ROOT/models/snlc_sim.defaults  ,
```

where xxx refers to the Ia model that is chosen. Since the extinction (A_V) profile is exponential instead of Gaussian, the A_V profile is controlled by the parameters GENRANGE_AV and GENTAU_AV, where the latter is the exponential slope. You can change any default parameter by adding it in your private sim-input file or using the command-line override. Note that there are two GENSIGMA_xxx values to specify a bifurcated Gaussian with peak at GENMEAN_xxx. The GENRANGE_xxx specify parameter ranges to avoid extreme or unphysical values.

Aside from the profile-parameters above, the SN model parameters from the training reside in

```
$SNDATA_ROOT/models/[mlcs2k2,stretch,SALT2_templates]
```

For mlcs2k2, you can specify the default with "GENMODEL: mlcs2k2", or specify any option in the /mlcs2k2 subdirectory. The SALT2 templates are fixed as of this version, but options may be available later. For the stretch and stretch2 models you must specify a template in the sim-input file with

```
STRETCH_TEMPLATE_FILE: <mytemplate.dat>
```

The simulation will first check if this file exists in $SNDATA_ROOT/models/stretch; if not, then the simulation will check your current working directory. If the template file cannot be found, the simulation will abort.

You can adjust the rise-time for any model using the following sim-input parameters:

```
GENMEAN_RISETIME_SHIFT:  2.3      # shift at -18 days
GENSIGMA_RISETIME_SHIFT: 0.6 0.6
GENRANGE_RISETIME_SHIFT:  -4. 4.
```

In the above example, the rest-frame rise-time at each epoch is increased by $2.3 \times T_{\rm rest}/18$ days with a Gaussian sigma of 0.6 days, and shifts past ± 4 days are excluded. The rise-time adjustments can be used, for example, to generate a double-stretch model by simulating two separate samples, each with a different rise-time shift.

Finally, select the Ia integer-type-code with input key "SNTYPE_Ia" (default Ia type is 1). For example, the SDSS–II code for type Ia is

```
SNTYPE_Ia: 120
```

and this code appears in the header of each output data file after the "SNTYPE:" key.

3.4 Simulating Non-Ia Supernova (II, Ib, Ic)

While the Type Ia light curve models are based on a parametric equation or photometric templates, the non-Ia models are based on smoothed light curves and spectral templates. A spectral template can be a smoothed average from an ensemble of observations, or a template can correspond to a particular (well-observed) supernova where a composite spectrum is warped to match the observed photometric colors. The `snlc_sim` simulation is designed to take full advantage of both types of templates. In addition to non-Ia SNe, this feature can be used to simulate peculiar Ia, theoretical models, AGN, or any transient object.

There are two methods for generating non-Ia light curves

```
GENMODEL:  nonIa  # (or non1a) rest-frame mag + K-correction
GENMODEL:  NONIa  # (or NON1a) compute observer-mag from SED (like SALT2)
```

The rest-frame "`nonIa`" model is useful if the SED has not been warped to match the photometry of the corresponding light curve, or if you want to simulate host-galaxy extinction using CCM89 (i.e., specifying R_V and an A_V distribution). A potential disadvantage of this method is the need to generate many K-correction tables. The observer-frame "`NONIA`" model is useful if each SED has already been warped to match the photometry of the underlying light curve. The observer-frame magnitudes are computed directly from the SED the same way as for SALT2, and in fact using the same software tools. Since no K-corrections are needed, the user can point to the SN Ia K-corr table to read in the filters and primary reference. For the remainder of this section, the instructions apply to both the `nonIa` and `NONIa` options, except for brief mentions of `nonIa` K-correction tables.

A list of available non-Ia templates is given in

```
$SNDATA_ROOT/snsed/non1a/NON1A.LIST
```

As explained below, the user selects non-Ia templates using the integer indices in the `NON1A.LIST` file. For the rest-frame `nonIa` option, the corresponding K-corrections are in `$SNDATA_ROOT/kcor/NON1A`. As more non-Ia templates become available, the `NON1A.LIST` file and K-corrections will be updated by the relevant experts. The simplest way to select non-Ia from the sim-input file is as follows,

```
GENMODEL:   non1a
NON1A:      0  1.  # index and wgt
```

where specifying an `INDEX` of zero is an instruction to use all available non-Ia (in the `NON1A.LIST` file) with equal weight. Inside each generated light curve file, the non-Ia `INDEX` is specified by "`SIM_NON1A: INDEX`"; the corresponding `SNANA` variable is `SIM_NON1A(isn)`, and the fitres-ntuple (ntid 7788) variable is "`non1a`." Although this "all" option is convenient to quickly study Ia-contamination using all available non-Ia templates, it is not very convenient for normalizing the individual types.

To specify normalizations and simulation parameters that depend on the non1a type,

```
NGENTOT_LC: 1000
GENMODEL:   nonIa  # or NONIa or non1a or NON1a
NON1A_KEYS: 5 INDEX  WGT  MAGOFF  MAGSMEAR  SNTYPE
```

```
NON1A:         2     0.2   0.0      1.2       2
NON1A:         3     0.2  -0.2      0.8       2
NON1A:         4     0.4  -0.6      0.9       3
```

Only the GENMODEL argument distinguishes upper and lower case; the other keys all use upper-case NON1A. Since the weights (WGT) are re-normalized to sum to unity, the first 1/4 of the generated SNe will use INDEX = 2, the second 1/4 will use INDEX = 3, and the latter 1/2 will use INDEX = 4. The keyword NGENTOT_LC specifies the total number to generate, which is different from NGEN_LC that specifies the number passing trigger & cuts and that are written out to SNDATA files. The relative WGT factors make physical sense only if NGENTOT_LC is used to specify the statistics.

The NON1A_KEYS are used to specify additional parameters that depend on non1a type. MAGOFF is a global magnitude offset applied to all passbands (i.e, equivalent to the GENMAG_OFF_MODEL keyword), and is used to adjust the average brightness of each non1a type. For SEDs that are normalized to have a peak mag of zero (i.e., the Nugent templates) rather than to physical units (erg/s/Å/cm^2), MAGOFF must be used for the NONIA option to get meaningful results. For the nonIa model the SEDs are used only for K-corrections, and therefore the SED normalization does not matter. MAGSMEAR is a Gaussian σ for global coherent magnitude fluctuations (i.e, equivalent to GENMAG_SMEAR), and is used to simulate intrinsic variations. SNTYPE is a spectroscopic typing-index that appears after the SNTYPE keyword in the SNDATA files.[1] In the above example, non-Ia indices 2 & 3 are both type II SNe, so they both get SNTYPE=2. Non-Ia index 3 is a different SN type, so it gets a different SNTYPE value. SNTYPE=1 is always reserved for type Ia.

Additional parameters can be added to the software as needed. With no keys specified, the default is two keys: INDEX & WGT. This default ensures that old sim-input files (i.e, from SNANA v8_16 and earlier) will work. Also note that INDEX & WGT are required to be the first two keys in the list; if not, the simulation will abort with an error message.

In this example, the end of the README file will show the statistics for each non-Ia INDEX as follows:

```
NON1A-SUMMARY vs. INDEX:
            NGEN    NGEN   SEARCH                Rest Peakmag+19
INDEX(TYPE) written total  Effic    CID-range      a     b     c     d     e
-------------------------------------------------------------------------------
002 (  IIn)   132    333   0.396      1 - 40333   1.70  1.87  2.11  2.30  2.52
004 (  IIL)    95    667   0.142  40334 - 41000   2.88  1.99  1.88  2.11  2.13
```

Note that the column under "NGEN total" sums to the requested number of generated SNe (1000), and the column under "NGEN written" shows how many SNe pass trigger & selection cuts and were thus written to SNDATA files. The Search-Effic, CID-range and Rest-Peakmags are printed for convenience. SNe with the same INDEX are generated sequentially, and then the next INDEX is generated; *i.e., the non-Ia indices are NOT randomly mixed.* The sequential generation by INDEX is done for speed, and avoids re-initializing templates multiple times. It is therefore crucial to analyze the *entire* simulated sample in order to have the correct mixture of non1a types.

[1] You can set the Ia type with input key "SNTYPE_Ia" (default Ia type is 1)

Finally, a few comments about "nonIa "K-corrections. If you do NOT specify a KCOR_FILE in the sim-input file, then the default K-correction file will be used automatically for each non-Ia INDEX. If you specify a KCOR_FILE, then this K-correction file will be used for all non-Ia indices. The user-KCOR_FILE is useful, for example, to study the sensitivity to K-corrections. For the NONIA option you must specify a KCOR_FILE in the sim-input file, but you can use the same one used for simulating SNe Ia.

3.4.1 Private $NON1A_ROOT

Instead of using the public $SNDATA_ROOT for the non-Ia templates, a private "$NON1A_ROOT" area can be used for developing non-Ia templates, or using proprietary templates. This $NON1A_ROOT directory is essentially a duplicate of $SNDATA_ROOT, but includes only the non-Ia directories. Depending on whether you choose the "NON1A" or "non1a" model, the required private directories are

```
mkdir $NON1A_ROOT/snsed/NON1A   # for ''GENMODEL: NON1A''

mkdir $NON1A_ROOT/snsed/non1a   # for ''GENMODEL: non1a''
mkdir $NON1A_ROOT/kcor/non1a    # idem
mkdir $NON1A_ROOT/filters       # idem
mkdir $NON1A_ROOT/standards     # idem
```

The NON1A model needs only one directory containing the SEDs, while the non1a model needs more directories to generate and store the K-correction tables. For the non1a model, the K-correction tables must be generated using the snana script "$SNANA_DIR/util/kcor_gen_non1a.cmd," where the usage instructions are at the top of the file. This script allows the simulation to automatically associate each non-Ia SED with the corresponding K-correction table.

You can specify $NON1A_ROOT either by setting an environment variable in your session, or by adding the sim-input key

```
NON1A_ROOT:  /my_private_non1a_dir
```

The sim-input key above takes priority over the value of the environment variable.

3.5 The 'SIMLIB' Observing Conditions File

The cadence, CCD properties and seeing conditions are defined in a 'SIMLIB' file, meaning "simulation library." As an example, the start of the SIMLIB file for the SDSS-II 2005 survey is shown in Fig. 1. The public SIMLIB files are located in $SNDATA_ROOT/simlib, and you specify a SIMLIB file in your input file with the keyword

```
SIMLIB_FILE: SDSS2005_ugriz.SIMLIB
```

The simulation will first check YOUR current working directory for this file; if it's not there, then snlc_sim will check the public directory $SNDATA_ROOT/simlib.

A SIMLIB file is created by someone with knowledge of the conditions. There is a convenient utility, SNANA_DIR/src/simlib_tools.c, that you can use to create the SIMLIB file in the correct format. This utility has a lot of error checking to prevent you from accidentally writing absurd values like a negative PSF. In principle, a SIMLIB file need be created only once per survey, although systematic studies may result in many SIMLIBs. If there are several exposures per filter per night, the utility simlib_coadd.exe (§7.6) will re-make a SIMLIB with all exposures per filter combined into a single effective exposure per night.

For a non-overlapping field, the "FIELD:" header should appear only once per MJD-sequence. For overlapping fields, the FIELD key appears more than once as indicated in Fig. 1. You can repeatedly toggle between two fields, or simply list all the MJDs for one field (i.e, 82N) followed by all of the MJDs for the other field (i.e., 82S); the MJDs need not be chronological in the SIMLIB, as the simulation will sort them internally. You can ignore the FIELD key in the SIMLIB as well, in which case the SNDATA files and analysis lose track of the field.

The simlib header values for RA,DECL have units of degrees, the PIXSIZE value is in arcseconds, and MWEBV is the $B-V$ color excess due to Galactic extinction. If MWEBV is zero, this is a flag for the simulation to use the default dust map from [6].

Here is a brief explanation for each column in the SIMLIB file:

1. **S: or T:** refers to Search (required) or Template (optional). Template information is used only to add more skynoise that would result from subtraction. Instead of including template info, you could simply increase the Search-image skynoise by a small amount.

2. **IDEXPT:** arbitrary identifier. You can set this to zero if you want. For SDSS, it glues together the run and field numbers.

3. **FLT:** single character to specify a filter for this observation.

4. **CCD Gain:** Number of photo-electrons per ADU or DN.

5. **CCD Noise:** CCD read noise in photo-electrons, per pixel. This term is usually much smaller than the SKYSIG term below.

6. **SKYSIG:** Standard deviation of sky, in ADU (or DN) per pixel. See §3.7 for noise calculation. You can optionally enter the skysig values per arcsec^2 by specifying the simlib header key

   ```
   SKYSIG_UNIT: ADU_PER_SQARCSEC
   ```

   The simulated README file includes the SKYSIG_UNIT value.

7. **PSF1,2 and RATIO:** The PSF is specified by a double-Gaussian with σ-widths (pixels) of PSF1 and PSF2. The ratio refers to PSF2(origin)/PSF1(origin). Note that the default PSF unit is in pixels, not arcsec. You can optionally give the PSF values in the more astronomy-friendly units of arcsec-FWHM by specifying the simlib header key

   ```
   PSF_UNIT:  ARCSEC_FWHM
   ```

The simulated README file includes the PSF_UNIT value.

8. **ZPTAVG:** Zero point relating the source brightness to the CCD flux measured in ADU:

$$\mathrm{Flux(ADU)} = 10^{-0.4(m-\mathrm{ZPTAVG})}$$

where m is the magnitude of the source. For example, if F_{20} is the flux (in ADU) for a point source with mag$= 20$, then **ZPTAVG**$= 20 + 2.5\log_{10}(F_{20})$. Note that **ZPTAVG** encodes information about the atmospheric transparency, telescope aperture & efficiency, and the exposure time. For any given simlib file, the simulated **ZPTAVG** can be changed globally or by filter as explained in §3.13.

9. **ZPTSIG:** Standard deviation among stars used to get zeropoint. This 'sigma' is used to add additional smearing to the magnitude and calibrated flux.

A sequence of MJDs that span the survey constitutes one entry in the SIMLIB, and the index "LIBID" labels each entry. A SIMLIB can have one LIBID, or hundreds. Large-area surveys, like SDSS-II, need hundreds of LIBIDs to properly sample the sky. A small area survey, like DES, may need just one LIBID per pointing, and per season.

3.6 Simulating Overlapping Fields

The simulation handles overlapping fields, such as for the SDSS **82N/82S** overlap, and the 20% overlap of the LSST fields. Overlapping fields are specified in the SIMLIB file by simply specifying the FIELD keyword as needed. Figure 1 above illustrates an overlap between the SDSS fields **82N** and **82S**. Note that overlapping fields make no sense if there is just one simlib entry per field with the position selected at the (non-overlapping) center of the field. The use of overlapping fields should be used with many simlib entries per field, and using random (or grid-spaced) coordinates that probe both the overlapping and non-overlapping regions.

By default all fields are analyzed when running the fitting program. However, specific fields can be selected using the &SNLCINP namelist variable SNFIELD_LIST.

```
SURVEY: SDSS     FILTERS: gri
USER: rkessler     HOST: sdssdp47.fnal.gov
COMMENT: 'Extract random RA,DECL,MJD from MYSQL: year=2005'
BEGIN LIBGEN  Tue Apr 17 13:32:33 2007

# ----------------------------------------
LIBID: 1
RA: 26.430172    DECL: 0.844033   NOBS: 42    MWEBV: 0.026   PIXSIZE: 0.400
FIELD: 82N

#                           CCD  CCD         PSF1 PSF2 PSF2/1
#     MJD      IDEXPT  FLT GAIN NOISE SKYSIG (pixels)  RATIO  ZPTAVG ZPTSIG
S: 53616.383  556600405 g  4.05  4.25  4.04  1.85 3.61 0.247  28.36  0.020
S: 53616.383  556600405 r  4.72  4.25  5.28  1.64 3.62 0.142  28.17  0.022
S: 53616.383  556600405 i  4.64 12.99  6.95  1.60 3.81 0.103  27.84  0.017
FIELD: 82S
S: 53622.395  558200552 g  4.03  5.45  4.09  1.58 3.31 0.107  28.45  0.018
S: 53622.395  558200552 r  4.89  4.65  5.00  1.46 3.55 0.065  28.15  0.028
S: 53622.395  558200552 i  4.76 10.71  6.43  1.53 3.65 0.075  27.85  0.029
S: 53626.359  560300625 g  4.05  4.25  4.40  1.83 3.50 0.282  28.24  0.020
etc ...
```

Figure 1: Header and part of first entry of the SIMLIB file used for the SDSS-II survey.

3.7 Noise Calculation

Here is an analytic calculation of the noise from the signal and sky-background. The error on the signal (in photoelectrons) is simply $\sqrt{\mathcal{N}_{\rm pe}}$. To get the error in observed CCD counts (ADU), we start by relating the signal counts in ADU to the number of photoelectrons,

$$\mathcal{N}_{\rm ADU} = \mathcal{N}_{\rm pe} \times G^{-1} \times \mathcal{S}_{ZP} \tag{1}$$

where $\mathcal{N}_{\rm pe}$ is the number of observed photoelectrons, G is the number of photoelectrons per ADU (CCD gain), and $\mathcal{S}_{ZP}$ is a scale factor applied to the signal so that the SN magnitude is referenced to the template zeropoint. This factor is $\mathcal{S}_{ZP} = 10^{0.4(ZP_t - ZP_s)}$, where $ZP_{s,t}$ are the zeropoints for the search and template runs. In cloudy conditions, $\mathcal{S}_{ZP} >> 1$ because the signal is much smaller than it would have been in the template run. If no template is given, then $\mathcal{S}_{ZP} = 1$. The error on the signal (in ADU) is

$$\sigma^2(\mathcal{N}_{\rm ADU}) = \mathcal{N}_{\rm ADU} \times G^{-1} \times \mathcal{S}_{ZP} \tag{2}$$

The sky-background error is computed from

$$\sigma_{skytot} = \mathcal{S}_{ZP} \times \sqrt{A \times (\sigma^2_{skypix} + \bar{\sigma}^2_{skypix})} \tag{3}$$

where σ_{skypix} is the search-run skynoise per pixel, $\bar{\sigma}_{skypix}$ is the template-run skynoise, A is the effective area in square-pixels, and the units are ADU. There is a similar term for the CCD readout noise per pixel (summed over the area), but it is left out here in this example.

Using double-Gaussian fit parameters for the PSF, in which $\sigma_{1,2}$ are the PSF-sigma for the two Gaussians, and $h_{1,2}$ are the heights at the origin[2], the effective area is

$$A = \frac{1}{\int PSF^2(r,\theta) r dr d\theta} = \frac{4\pi(\sigma_1^2 + \sigma_2^2)}{1 + 4\pi^2\sigma_1^2\sigma_2^2(h_1+h_2)^2} = 4\pi\sigma_1^2 \left[\frac{(1+R_\sigma^2)(1+R_\sigma^2 r_h)^2}{R_\sigma^2(1+r_h)^2 + (1+R_\sigma^2 r_h)^2} \right] , \tag{4}$$

where in the second step $R_\sigma \equiv \sigma_2/\sigma_1$ and $r_h \equiv h_2/h_1$. For a single-Gaussian PSF, $r_h \to 0$ for any value of R_σ, and the area is just $4\pi\sigma^2$. For SDSS data, typical second Gaussian parameters are $R_\sigma = 2$ and $r_h \sim 0.05$; this increases the area to $\sim 1.23 \times 4\pi\sigma_1^2$, and increases the sky-noise estimate by $\sim 10\%$.

The rest of this section will perform an explicit calculation of the noise, using an example from a DES simulation. Here is the information for a random epoch on a random simulated lightcurve:

[2] A double-Gaussian PSF with normalization $\int PSF(r,\theta) r dr d\theta = 1$ has $2\pi(\sigma_1^2 h_1 + \sigma_2^2 h_2) = 1$

PASSBAND:	g	r	i	z	Y	
GAIN:	1.000	1.000	1.000	1.000	1.000	e/ADU
RDNOISE:	10.000	10.000	10.000	10.000	10.000	e-
SKY_SIG:	108.81	105.36	217.58	378.84	848.53	ADU
PSF_SIG1:	1.500	1.500	1.500	1.500	1.500	pixels
FLUX:	14707.89	12515.67	30756.55	28334.23	62748.97	ADU
FLUX_ERRTOT:	590.695	571.406	1170.485	2022.083	4518.783	ADU
FLUXCAL:	18.52	42.21	46.13	46.59	51.71	(x10^11)
FLUXCAL_ERRTOT:	0.986	2.403	2.385	3.683	4.141	
MAG:	24.3311	23.4364	23.3402	23.3292	23.2160	
MAG_ERRPLUS:	0.0569	0.0624	0.0565	0.0898	0.0892	
MAG_ERRMINUS:	0.0569	0.0624	0.0565	0.0898	0.0892	
ZEROPT:	34.7500	33.6800	34.5600	34.4600	35.2100	
ZEROPT_SIG:	0.0350	0.0340	0.0350	0.0340	0.0350	

For simplicity, note that the PSF is modeled as a single-Gaussian, and that the gain is unity. Now let's compute that flux and noise for *i*-band.

The measured flux (in ADU) and the calibrated flux (for lightcurve fits) are given in terms if the magnitude (m_i) and zeropoint (Z_i),

$$\texttt{FLUX} = 10^{-0.4(m_i - Z_i)} = 10^{-0.4(23.3402 - 34.56)} = 30755 \text{ ADU} \quad (5)$$

$$\texttt{FLUXCAL} = 10^{-0.4 m_i} \times 10^{11} = 46.13 \quad (6)$$

which agrees with the simulated values above. Since the gain is unity, 1 ADU = 1 photoelectron (p.e.), and the noise from signal-photostatistics is $\sigma_{sig} = \sqrt{N_{pe}} = 175.4\ p.e.$.

To include the sky-noise, we use the following information from the simulation library (see above dump): SKYSIG= 217.58 ADU/pixel, PSF(σ) $= 1.50\sqrt{pixels}$, and 1 pixel is $0.27'' \times 0.27''$. The effective aperture area is $A = 4\pi \times \text{PSF}^2 = 28.27$ pixels. The total skynoise is thus $\sigma_{sky} = \texttt{SKYSIG} \times \sqrt{A} = 1156.95\ p.e.$ The total noise on the flux is $\texttt{FLUX\_ERRTOT} = \sqrt{\sigma_{sig}^2 + \sigma_{sky}^2} = \sqrt{1156.94^2 + 175.4^2} = 1170.2\ p.e = 1170.2$ ADU. The signal-to-noise ratio (SNR) is $30755/1170 \simeq 26$.

3.8 K-corrections

For the stretch and MLCS2k2 models, K-corrections are needed in both the simulation and the fitter. K-corrections are applied using a technique very similar to that used in [7, 1]. The basic idea is that for each epoch and each pass-band, the spectral template is warped by applying the CCM89 extinction law with a variable A_V; "A_V-warp" is the value of A_V for which the synthetic color matches the color of the rest-frame lightcurve. The K-correction is then determined from this A_V-warped spectral template. Note that this method is model-independent and can therefore be applied to any lightcurve model.

The K-corrections and synthetic magnitudes are NOT computed internally because this would result in slow code, especially for the fitter. To speed up the calculations of these convolution-integrals, K-corrections and synthetic mags are read from a lookup table generated with SNANA_DIR/bin/kcor.exe. Before running the simulation or fitter, the program kcor.exe must run, although it runs once and only once until you need K-corrections that are not defined. The kcor program reads a self-documented input file such as

```
$SNDATA_ROOT/analysis/sample_input_files/kcor/kcor_[SURVEY].input
```

and then generates lookup tables as a function of redshift, epoch, and extinction parameter A_V. A typical table binning is 0.05 in redshift, 1 day in rest-frame epoch, and 0.25 in A_V; this binning is used to store every user-defined K_{xy}, and to store synthetic lightcurves for every user-defined filter.

A linear interpolation routine[3] determines a K-correction for arbitrary z,epoch,A_V. A special function, GET_AVWARP, finds the magic A_V-warp parameter such that the warped spectral template has the same color as your lightcurve. The table format is CERNLIB's HBOOK, and is stored in

```
$SNDATA_ROOT/kcor   .
```

The subroutines that read and interpolate the kcor tables are written in fortran, and are stored in snana.car. The SNANA product includes a fortran library ../lib/libsnana.a, which allows C programs to use the fortran utilities. The extern statements at the top of snlc_sim.c declare the fortran functions used to lookup K-corrections.

[3] a double precision version of CERNLIB's FINT is used for multi-dimensional linear interpolations.

3.9 Intrinsic Brightness Variations

There are four ways to introduce intrinsic variations that result in anomalous scatter in the Hubble diagram:

```
# ------------------------------------------------------
# method 1: coherent variation in all epochs & passbands
#           note: colors are not varied.
GENMAG_SMEAR: 0.1  # coherent mag-smear in all measurements

# method 2: independent variation in each passband (coherent among epochs)
#           that results in color variations
GENMODEL_ERRSCALE: 1.1    # scale MLCS peak-model error; apply smearing
GENMAG_SMEAR_FILTER: UBV 0.05 # and/or fixed smearing per filter
GENMAG_SMEAR_FILTER: RI  0.08 # and/or fixed smearing per filter

# method 3: wavelength-dependent sigma based on
#           lamrest = <LAMOBS>/(1+z).   Select file in current dir
#           or in $SNDATA_ROOT/$SNDATA_ROOT/models/modelSmear
MODELSMEAR_FILE:  colorSmear_SNLS3.dat

# method 4: vary RV with asymmetric Gaussian distribution
GENMEAN_RV:         1.6             # location of Gauss peak
GENSIGMA_RV:        0.1   0.9       # lower,upper Gaussian-sigmas
GENRANGE_RV:        1.3   4.5       # gen-range for RV
# ------------------------------------------------------
```

Note that methods 1&2 can be used together, as well as methods 1&3. However, methods 2&3 cannot be used together.

For rest-frame models, the argument of GENMAG_SMEAR_FILTER is a list of rest-frame filters; for observer-frame models, the argument is a list of observer-frame filters.

The file specified by MODELSMEAR_FILE contains a table of "wavelength sigma". This file is first searched in your current directory; if not there then the official area is searched as indicated above. If you fix the color and stretch parameter distributions to each be a delta function, set EXPOSURE_TIME$>>$ 1, and remove the default $\pm 2\sigma$ clipping requirement (by setting SIGMACLIP_MAGSMEAR: -999 +999) then the expected rms spread in the generated peak magnitudes (see MAGT0_[filter] in SIMGEN_DUMP option) should correspond to the intrinsic spread that is speficied. This agreement is readily obtained for SALT-II, but not quite for MLCS2k2. For MLCS2k2, K-corrections dilute the smearing because the two rest-frame filters used for color warping are each smeared by a different random number.

3.10 Search Efficiency

The simulation includes options to model the search efficiency of the survey. The search efficiency is broken into two parts: (i) image subtraction pipelines and (ii) human efficiency that includes visual scanning and spectroscopic targeting and selection. The defaults are usually set so that the full search efficiency is used, resulting in a simulated sample that best matches the spectroscopically confirmed SN Ia data sample. However, for special samples such as photometrically identified SNe Ia, one must be careful about selecting the appropriate search-efficiency options. Some explicit examples will be given at the end of this section. To ensure that your settings are correct, it is recommended to always check that the simulated redshift distribution matches that of the data.

The image subtraction pipeline efficiency, ε_{subtr}, is based on software algorithms and therefore in principle this part of the efficiency can be rigorously determined. The simplest specification of the search efficiency is based on requiring a minimum number of observations with the sim-input keyword "MINOBS_SEARCH: <MINOBS>". The simulation also allows for two detailed methods to determine ε_{subtr}. The first method is based on the SDSS method of inserting fake SNe Ia into the search images during the survey, and using this sample to measure the pipeline efficiency as a function of signal-to-noise (SNR) in each filter. The resulting information is stored in

```
$SNDATA_ROOT/models/searcheff/SEARCHEFF_SNR_SDSS-[gri].DAT ,
```

and one can replace "SDSS" with the name of any other survey that has similar information. The second method is based on measuring the efficiency as a function of magnitude: an example with HST data is in

```
$SNDATA_ROOT/models/searcheff/SEARCHEFF_MAG_HST-[01234567].DAT ,
```

and "HST" can be replaced with the name of any other survey that has similar information. Warning: do NOT define both the SNR-based and MAG-based efficiency for a survey.

The above information tells us if a given epoch is detected in a particular filter, but does not specify if the supernova would have been discovered. The *discovery* logic is defined for each survey in the file:

```
more  $SNDATA_ROOT/models/searcheff/SEARCH_PIPELINE_LOGIC
SDSS: 3 gr+ri+gi  # require 3 epochs, each with detection in two bands.
HST:  1 6         # require 1 epoch with detection in filter '6' = F850LP_ACS
```

where the first number is the minimum number of epochs required, and the second string defines the logic for a single-epoch detection. In the above examples, the SDSS discovery logic requires three epochs, where each epoch has a detection in at least two of the three *gri* filters. The HST discovery logic simply requires a single detection in filter '6'. If no SEARCHEFF_XXX files are defined for a survey, then ε_{subtr} is assumed to be one. If the SNR-based or MAG-based files exist, then these files are read and ε_{subtr} is determined for each simulated SN. The image-subtraction efficiency can be applied in the simulation, or the results can be stored in the data files for future analysis: the control flag APPLY_SEARCHEFF_OPT is discussed below after the human efficiency is discussed.

The human/spectroscopic efficiency (ε_{spec}) cannot be rigorously computed since it involves human decision making during the survey. This part of the efficiency can be estimated from photometrically identified SNe Ia, or by comparing data and simulated distributions. The simulation allows for ε_{spec} to be defined as an exponential function of redshift or "magdim" ($\mathcal{M}_{\rm dim}$), where $\mathcal{M}_{\rm dim}$ is the relative intrinsic brightness of a SN Ia. For MLCS2k2, $\mathcal{M}_{\rm dim}$ depends on Δ and A_V: for SALT-II, $\mathcal{M}_{\rm dim}$ depends on x_1 and c: The exponential parameters for all surveys are explained and defined in the file

```
$SNDATA_ROOT/models/searcheff/HUMAN_SEARCHEFF_V3.DAT .
```

The option is specified in the sim-input file with the keyword

```
HUMAN_SEARCHEFF_OPT: <opt>
```

For each simulated SN Ia, the search algorithm is evaluated separately for the image-subtraction and spectroscopic efficiencies. Random numbers are compared against the efficiencies to determine which epochs/SNe are selected. The results of these two search algorithms are stored in a bit-mask in each data file, with the following possibilities:

```
SIM_SEARCHEFF_MASK: 1   # detected by image-subtr; failed spec follow-up
SIM_SEARCHEFF_MASK: 2   # failed image-subtr; passed spec follow-up
SIM_SEARCHEFF_MASK: 3   # detected by both image-subtr & spec follow-up
```

Note that SIM_SEARCHEFF_MASK= 2 corresponds to an unphysical case since it is unlikely to get a spectrum of a SN that was not identified by the image-subtraction pipeline. Although the search efficiencies are always evaluated, the user has the option to apply these efficiencies in the simulation, or to write out all simulated SNe and use the SIM_SEARCHEFF_MASK for further investigation:

```
APPLY_SEARCHEFF_OPT: 0  # write SIM_SEARCHEFF_MASK only
APPLY_SEARCHEFF_OPT: 1  # reject SN based on search eff
```

When "APPLY_SEARCHEFF_OPT: 1" is set, then SIM_SEARCHEFF_MASK will always be set to 3 because the simulated SNe Ia that fail either one of the search criteria are rejected. It is recommended to always "apply" the search efficiency unless a specific study is aimed at rejected SNe Ia.

Here are a few examples of sim-input settings. First, to simulate a spectroscopically confirmed sample,

```
APPLY_SEARCHEFF_OPT: 1       # apply all search efficiencies
SOFTWARE_SEARCHEFF_OPT: 1    # evaluate image-subtr effic.
HUMAN_SEARCHEFF_OPT: <opt>   # expon function for spec effic
```

should result in a redshift distribution that matches that of the data. If all of the missed SNe Ia (i.e., those with no spectral confirmation) are photometrically identified, the appropriate settings for the simulation are

```
APPLY_SEARCHEFF_OPT: 1       # apply all search efficiencies
SOFTWARE_SEARCHEFF_OPT: 1    # evaluate image-subtr effic.
HUMAN_SEARCHEFF_OPT: 0       # ignore spec effic.
```

so that $\varepsilon_{spec} = 1$, but the image-subtraction efficiency is still applied.

3.11 Spectroscopic Typing

By default, the SNTYPE keyword is always filled in the simulated SNDATA files, corresponding to 100% spectroscopic typing. One can also simulate a sample with limited spectroscopic typing, which is particularly useful for classifying blinded samples. A spectroscopically typed subset is defined as follows:

```
# EFF = EFF0 * (1-x^EFFEXP),  x=(PKMAG-MAGMIN)/(MAGMAX-MAXMIN)

            FILT  EFF0 MAGMIN MAXMAG EFFEXP  ZERR
  SPECTYPE:   r   0.40   16.0  21.5    5    0.005
  SPECTYPE:   i   0.40   21.5  23.5    6    0.005
```

which gives the magnitude ranges and efficiency function ($\varepsilon_{spec} = \varepsilon_0(1-x^n)$) for each filter. The factors $1-x^{5,6}$ cause a sharp efficiency roll-off near the magnitude limit; by definition, $\varepsilon_{spec} = 100\%$ at MAGMIN and zero at MAGMAX. For a given filter, a spectroscopic type is given (i.e., SNTYPE) if the calculated ε_{spec} is larger than a random number between 0 and 1. The logical-OR is taken when multiple filters are specified. For a spectroscopically typed SN, the measured (i.e., smeared) REDSHIFT_FINAL is determined from ZERR above. For un-typed SNe, SNTYPE is set to -999. Unless the BLINDING option is selected (§3.21), the SIM_NON1a index always appears regardless of whether SNTYPE is set.

3.12 Selection Cuts

Although selection cuts are usually applied with the fitting program (§ 4) or some external program, the simulation allows for some basic selection cuts. This feature is particularly useful, for example, to simplify and speed up the generation of a large efficiency grid, and to estimate rates. The global flag to implement the "CUTWIN_XXXX" selection criteria is

```
APPLY_CUTWIN_OPT: 1  # => implement selection cuts
APPLY_CUTWIN_OPT: 0  # => ignore selection cuts (default)
```

and a list of available cut-commands are as follows:

```
EPCUTWIN_LAMREST: 3000 9500   # cut-window for <lamobs>/(1+z)
EPCUTWIN_SNRMIN:  +3 1E8      # min SNR for each observation
CUTWIN_TRESTMIN: -19  -5      # at least 1 epoch before -5 d (rest-frame)
CUTWIN_TRESTMAX: +30 +80      # at least 1 epoch past +30 d
CUTWIN_TGAPMAX:    0  20      # largest Trest gap (days)
CUTWIN_T0GAPMAX:   0  10      # largest Trest gap near peak (days)
CUTWIN_NOBSDIF:    6  999     # Number of obs passing MJDDIF cut
CUTWIN_MJDDIF:     0.4 999    # NOBSDIF++ if this much later than last MJD
CUTWIN_NEPOCH:     7  +5      # require 7 epochs with SNR>5
CUTWIN_SNRMAX: 10 griz 1 -20 60  # SNR>10 for at least 1 of griz filters
CUTWIN_SNRMAX: 5  griz 3 -20 60  # SNR>5  for at least 3 of griz filters
CUTWIN_SNRMAX: 5  griz 3   0 60  # idem, but after max
CUTWIN_SNRMAX: 5   ri  2 -20 60  # r & i must each have SNR > 5
```

Each cut-command can be specified in the sim-input file, or using the command-line override (§7.1) without the colon. Any CUTWIN_XXX that is not specified results in no cut. The cuts beginning with EPCUTWIN apply to every epoch (observation), and only measurements passing these EPCUTWIN_XXX requirements are used to evaluate cuts on global light curve properties. CUTWIN_NEPOCH includes its own SNR requirement; thus you could set "EPCUTWIN_SNRMIN: -5 1E8" so that all measurements, regardless of SNR, are used for cuts on TRESTMIN, TRESTMAX and TGAPMAX, while still requiring 7 observations to have SNR>5.

Multiple CUTWIN_SNRMAX requirements can be specified. Note that this cut requires a certain number of passbands to have a minimum SNR value, but does not specify which bands. For the example above, "CUTWIN_SNRMAX: 5 griz 3 -20 60" is satisfied if the maximum SNR is >5 for either *gri*, *grz*, *giz*, or *riz*. You can require SNR cuts for specific filters as illustrated above with "CUTWIN_SNRMAX: 5 ri 2 -20 60"; this requires both *r* and *i* to have a measurement with SNR>5.

The TGAPMAX requirement applies to observations between the lower-TRESTMIN and upper-TRESTMAX cuts; i.e., -19 to $+80$ days in the example above. The T0GAPMAX requirement applies to observations near peak, meaning that the gap must overlap the range between the upper-TRESTMIN and lower-TRESTMAX cuts; i.e., -5 to $+30$ days. In this example, a gap defined by -12 to -6 days is included in the evaluation of the TGAPMAX cut, but not in the T0GAPMAX cut. Gaps of -6 to $+2$ and $+20$ to $+35$ days are included in the evaluation of both cuts. A gap of $+8.0$ to $+85$ is ignored for both cuts.

The generation and cut-selection statistics are printed at the end of the sim-README file (§ 3.2). Here is an example when selection cuts are applied, while the search efficiency is not:

```
Generation Statistics:
       Generated   250 simulated light curves.
       Wrote       100 simulated light curves to SNDATA files.
Rejection Statistics:
           1 rejected by GEN-RANGE cuts.
           0 rejected by SEARCH-TRIGGER
         149 rejected by CUTWIN-SELECTION
SEARCH+CUTS Efficiency:  0.402 +-  0.031
```

An efficiency grid can be quickly computed by looping over the variables of interest (redshift, SN brightness, etc) and using `grep "SEARCH+CUTS"` to extract the efficiencies.

The number of generated events is specified by the keyword "NGEN_LC: 100", which instructs the simulation to generate 100 lightcurves that pass the SEARCH-TRIGGER & CUTWIN-SELECTION. To prevent an infinite loop when the efficiency is (accidentally) zero, specify

```
EFFERR_STOPGEN:   0.001  # stop sim when effic error is this small
```

so that the simulation will stop gracefully after generating 1,000 lightcurves that all fail cuts. To avoid unwanted program exits, make sure that the EFFERR_STOPGEN value is much smaller than the expected efficiency.

3.13 Varying the Exposure Time/Aperture/Efficiency

For testing future (i.e, non-existent) surveys, the exposure times can be varied relative to that of the SIMLIB, and avoids the need to create a new SIMLIB for each sequence of exposure times. The sim-input syntax is

```
EXPOSURE_TIME:    2.0          # global increase for all filters
EXPOSURE_TIME_FILTER: g 3.0    # x3.0 more exposure in g-band
EXPOSURE_TIME_FILTER: r 4.6    # x4.6 more exposure in r-band
```

Since the global EXPOSURE_TIME multiples the filter-dependent exposure times, the net exposure-time increase for the above example is 6 and 9.2 for *g* and *r*, respectively. The default exposure-time increase is one.

This option is equivalent running each exposure longer by the specified amount, or increasing the aperture or efficiency. Technically, the simulation does the following for each filter:

```
ZPT(SIMLIB)  -> ZPT + 2.5*LOG10(EXPOSURE_TIME)
SKYSIG       -> SKYSIG * sqrt(EXPOSURE_TIME)
CCDNOISE     -> CCDNOISE * sqrt(EXPOSURE_TIME)
```

3.14 Simulating Host PhotoZ

To use host-galaxy photoZs as a prior when doing SN-photoZ fits (§ 4.8), the simulation of host-galaxy photometric redshifts is based on an externally supplied "HOSTLIB" library that must reside in $SNDATA_ROOT/simlib. See existing *.HOSTLIB files for the required format. The host photoZ option is invoked with the keyword

```
HOSTLIB_FILE:  <name>  # must reside in $SNDATA_ROOT/simlib
```

Note that the redshifts need not be sorted, as the simulation will match the current redshift with the closest redshift in the library. Make sure that your HOSTLIB has adequate statistics over the redshift range of interest. For example, consider a too-small library with only 50 entries that spans $0 \leq z \leq 1$, with *z*-bins of 0.02; every simulated redshift with $0.49 < z < 0.51$ will use the same library entry at $z = 0.50$, resulting in pathological distributions based on this photoZ entry.

To ensure that the light-curve fitter cannot cheat when doing photoZ fits, there is an option to replace the output REDSHIFT_FINAL with the host-galaxy (photoZ) redshift so that the spectroscopic redshift is not available in the data file. This option is invoked by setting GENSIGMA_REDSHIFT to any negative number. To go a step further and do SN-only photoZ fits (no host) without any risk of cheating, simply set GENSIGMA_REDSHIFT to a large value like 0.05.

3.15 Simulating the SN Rate: Volumetric and per Season

To simulate a constant volumetric rate at all redshifts, include the following in your sim-input file,

```
DNDZ: HUBBLE
```

and to simulate a redshift-dependent rate that depends on a power of $1+z$,

```
DNDZ: POWERLAW  2.6E-5   1.5   # SN rate ~ (1+z)^1.5
```

Note that setting the second POWERLAW argument to zero is equivalent to the HUBBLE option of a constant rate. As a convenience, the output README file includes a dump of the SN volumetric rate in redshift bins of 0.1 (grep "MODEL-RATE"). The generated redshift distribution can be reweighted relative to the DNDZ option above using

```
DNDZ_ZEXP_REWGT:  -2.0             # DNDZ *= 1/z^2
   or
DNDZ_ZPOLY_REWGT:  1.0 -0.2 0.003  # DNDZ *= [1 - 0.2*z + 0.003*z^2]
```

The example with "DNDZ_ZEXP_REWGT: -2" will give a roughly flat redshift distribution. The second option allows the user to multiply the "DNDZ" redshift distribution by an arbitrary 2nd-order polynomial function of the redshift.

As a convenience, the absolute number of SN per season within your survey ($N_{\rm season}$) is written into the output README file as follows:

```
      Number of SN per season = 12345
```

This value does not depend on NGEN_LC or NGETOT_LC,[4] and it is not used in the simulation. This calculated value depends on the MJD range (GENRANGE_PEAKMJD), redshift range (GENRANGE_REDSHIFT), DNDZ option above, and coordinate ranges (GENRANGE_RA and GENRANGE_DECL). The sky area specified by the RA and DECL ranges can be overwritten by explicitly defining a solid angle in your sim-input file using

```
SOLID_ANGLE: 0.0204  # solid angle (steridian) for SN/season estimate
```

The SOLID_ANGLE option is useful when the survey consists of several dis-connected patches of sky, thereby requiring the RA and DECL ranges to represent a solid angle that is much larger than that of the survey. $N_{\rm season}$ can be used generate an arbitrary number of SN seasons. For example, to simulate 3 seasons set the following:

```
NGENTOT_LC:  37035   # 3*12345 = 3 seasons
```

[4] NGEN_LC is the number of SNe generated after trigger cuts and NGETOT_LC is the total number generated regardless of the trigger and cuts.

3.16 "Perfect" Simulations

To make detailed numerical crosschecks, there is a "perfect" option to simulate light curves with $\times 10^4$ nominal photostatistics, no galactic extinction (Milky Way and host), and no intrinsic mag-smearing. The light curve fitter should determine the shape and color parameters with very high precision, and the cosmology fitter should determine cosmological parameters that agree well with the input. This option is invoked with

```
GENPERFECT: 1
```

and it automatically overrides the relevant parameters so that you need not change your sim-input file. The top of the sim-`README` file summarizes the modified quantities. You can also unselect some of the "PERFECT" options by specifying a bit-mask as the `GENPERFECT` argument. To see the bit-mask options,

```
snlc_sim.exe  mysim.input  GENPERFECT -1
```

will list the current bit-mask options and then quit without generating any SNe. You can then run, for example,

```
snlc_sim.exe  mysim.input  GENPERFECT 6  # = 2+4 (bits 1 & 2)
```

which selects the $\times 10^4$ exposure-time option (bit 1) and turns off intrinsic mag-smearing (bit 2), but leaves Galactic and host-galaxy extinction as defined in your sim-input file.

3.17 Redshift-Dependent Parameters

Although the default simulation parameters are independent of redshift, you can specify an arbitrary z-dependence for SN-related parameters such as dust parameters R_V & τ_V, SALT-II parameters α & β, and the mean luminosity parameter for any model.

The z-dependence is specified as an additive shift. If the function is simple, you can specify a polynomial function up to 3rd order. For more complex functions you can specify the function explicitly in redshift bins. Your function must give a shift of zero at $z = 0$ so that the sim-input parameters are clearly defined at $z = 0$. Examples for specifying both types of parameter-shift functions are given in this file,

```
$SNDATA_ROOT/analysis/sample_input_files/SALT2/SIM_ZVARIATION.PAR  .
```

To get a complete list of parameters that can have a z-dependence, type

```
> snlc_sim.exe  mysim.input  ZVARIATION_FILE 0
```

Next, copy the `SIM_ZVARIATION.PAR` above to your working area, and modify as desired. Then add the following keyword to your sim-input file,

```
ZVARIATION_FILE: SIM_ZVARIATION.PAR
```

or use the command-line override (§7.1). You can also change the name of the "`ZVARIATION`" file.

3.18 Generating Efficiency Maps (for `MLCS2k2` Prior)

Efficiency maps needed by `MLCS2k2` can be created with the command

```
SIMEFF_MAPGEN.cmd  <simeff-input file>
```

and examples of the simeff-input file are in

```
$SNDATA_ROOT/analysis/sample_input_files/simeff_mapgen/
```

Since the generation of a 4-dimensional (z, A_V, Δ, R_V) efficiency map can be CPU intensive, this script distributes jobs on several nodes defined by the user, and it runs the simulation in a mode where there are no output files, and hence no secondary `SNANA` jobs are needed. Your sim-input file (specified inside the simeff-input file) must apply selection cuts as described in § 3.12. It is assumed that the selection efficiency does not depend on the result of the light curve fit.

When the simeff-jobs have finished, use 'cat' to combined all the files. In your fitter namelist, "`SIMEFF_FILE`" specifies the efficiency map.

3.19 Light Curve Output Formats

Each simulated light curve is written out to one "SNDATA" file in the directory

```
$SNDATA_ROOT/SIM/[GENVERSION]/[GENVERSION]_SN######.DAT  ,
```

where GENVERSION is the user-supplied version name, and "######" is a six digit identifier. The output format is controlled by the sim-input keyword GEN_SNDATA_SIM, and the various options are described below. Note that GEN_SNDATA_SIM is a bit-mask so that multiple formats can be included in each light curve file.

3.19.1 Default (verbose) Light Curve Output

By default, GEN_SNDATA_SIM = 1 and results in the standard verbose output used by SNANA and the fitting program.

3.19.2 Terse (non-verbose) Light Curve Output

If you want to use a non-SNANA program to analyze light curves simulated with SNANA, there is an option to get a simplified one-line-per-observation output for the light curves: "GEN_SNDATA_SIM: 2". By default, this option is set to 1 for the normal verbose output. With option 2, you still get the header info that includes header info like RA, DECL, and redshift, but the output is simplified to one line per measurement and may be easier to parse:

```
# ==========================================
# TERSE LIGHT CURVE OUTPUT:
#
NVAR: 9
VARLIST:  MJD  FLT FIELD  FLUXCAL  FLUXCALERR   SNR    MAG     MAGERR  SIM_MAG
OBS: 49562.316  Y 1694  -1.333e+03  5.891e+02  -2.26  128.000    0.000  27.313
OBS: 49572.430  z 1694   6.500e+01  1.708e+02   0.38   26.628  101.372  25.385
OBS: 49590.422  Y 1694   1.492e+02  1.635e+02   0.91   24.236  103.764  24.064
OBS: 49591.387  i 1694   3.733e+03  4.644e+02   8.04   23.970    0.144  24.198
OBS: 49591.414  z 1694   1.644e+03  3.271e+02   5.03   23.850    0.241  24.200
...
```

It is recommended to use the fluxes instead of mags because the mags are not defined for negative fluxes, and are ill-defined for very small fluxes. The FIELD is given for each measurement to properly label overlapping fields, SNR is the signal-to-noise ratio (FLUXCAL/FLUXCALERR) and SIM_MAG is the exact magnitude (without noise fluctuations) computed from the SN model.

To get both the verbose and terse formats, note that GEN_SNDATA_SIM is actually a bit-mask; therefore, specifying a value of 3 results in both outputs in each SNDATA file. Beware that snana.exe and snlc_fit.exe only read the verbose format.

3.19.3 Model-Mag Light Curve Output

To dump out the model mag info, set "GEN_SNDATA_SIM: 4" or set to 5 to get both the nominal output and the model-mag output. A sample output is as follows:

```
# ========================================
# MODEL MAG OUTPUT:
#
NVAR: 7
VARLIST: TOBS  FLT   MAGOBS  MAGERR  MAGREST   KCOR(SYM,VAL)
OBS:    -1.328  u    21.547   0.055  -19.810   K_Uu   1.379
OBS:    -1.328  g    20.228   0.035  -19.396   K_Bg  -0.205
OBS:    -1.328  r    20.182   0.037  -19.421   K_Vr  -0.157
OBS:    -1.328  i    20.335   0.047  -19.377   K_Ri   0.012
OBS:    -1.328  z    20.684   0.056  -19.237   K_Iz   0.268
etc ...
```

3.20 Simulation Dump Options

3.20.1 SIMLIB_DUMP Utility

To quickly check a SIMLIB, a screen-dump summary is obtained with the command:

```
 >  snlc_sim.exe  mysim.input  SIMLIB_DUMP 0
************************************************
  SIMLIB_DUMP

 LIBID  MJD-range    NEPOCH(all,gri) GAPMAX(frac) <GAP>
-------------------------------------------------------------
 001   53616-53705    126,42 42 42    11.0(0.12)  2.2
 002   53622-53700     51,17 17 17    19.0(0.24)  4.9
 003   53622-53705     69,23 23 23    10.1(0.12)  3.8
 004   53622-53705     54,18 18 18    15.0(0.18)  4.9
 ...
 050   53622-53705     57,19 19 19    15.0(0.18)  4.6

Done reading 496 SIMLIB entries.

LIBRARY AVERAGES PER FILTER:
              <PSF>
     <ZPT-pe> FWHM  <SKYSIG>  <SKYMAG>                Cadence
 FLT  (mag)  (asec) (ADU/pix) (asec^-2) <m5sig> <Nep>   FoM
-------------------------------------------------------------
  u   30.86   0.866       9.0   22.65    19.99   2.86   0.036
  g   34.31   0.828     117.8   20.75    19.99  29.71   0.123
  r   35.04   0.820     173.5   20.50    19.99  29.71   0.130
  i   34.81   0.829     216.2   19.74    19.99  31.43   0.128
  z   34.18   0.823     205.7   19.20    19.99  32.14   0.135
  Y   32.93   0.822     197.9   17.99    19.99  30.14   0.127

LIBRARY MIN-MAX RANGES:
        RA:    -59.994  to    58.766 deg
        DECL:   -1.253  to     1.257 deg
        MJD:   53616.2  to   53705.4

 CUT-WARNING:   46 SIMLIBS will fail user-cut on 'RA'
```

GAPMAX and <GAP> are the maximum and average gaps (days) between epochs in the SIMLIB. The "frac" after GAPMAX is the fraction of the MJD-range consumed by the largest gap. The LIBRARY MIN-MAX RANGES show you the ranges needed to include all SIMLIB entries. The CUT-WARNINGs shows

how many SIMLIB entries are excluded by the selection ranges in your sim-input file. CUT-WARNINGs are checked for RA, DECL and PEAKMJD.

In addition to the screen-dump, a one-line summary for each LIBID is written to [simlib].DUMP where [simlib] is the name of the SIMLIB file that you specify. This file is self-documented like the "fitres" files, and can be converted into an ntuple using the "combine_fitres.exe" program (§7.3).

3.20.2 Cadence Figure of Merit Utility

A figure of merit for the cadence can be determined in two ways. First, you can extract the function "SNcadenceFoM" from sntools.c and pass the arguments from your own wrapper function. The second method is to simply analyze any SIMLIB using the SIMLIB_DUMP option (§ 3.20.1). The FoM is appended to each entry in the output [simlib].DUMP file. The FoM for each simlib entry is a function of the MJD and the 5σ limiting magnitude for each observation.

3.20.3 SIMGEN_DUMP File

To quickly analyze generated distributions, there is an option to dump generated quantities to a column-formatted text file. For example, to check the generated redshift and SALT-II parameters, add the following to your sim-input file:

```
SIMGEN_DUMP: 5  CID  Z  S2x0 S2x1 S2c
   or
snlc_sim.exe mysim.input  SIMGEN_DUMP 5  CID Z S2x0 S2x1 S2c
```

which produces an auxiliary file [VERSION].DUMP in the same directory as the SNDATA files. The self-documented dump-file looks like:

```
NVAR: 4
VARNAMES:  Z S2x0 S2x1 S2c
SN:  50001  1.3820e-01 3.8970e-04 1.0906e+00 -1.5431e-01
SN:  50002  3.1260e-01 1.0278e-04 1.8671e-01 -3.9044e-01
SN:  50003  2.4248e-01 1.9417e-04 8.8190e-01 -3.8711e-01
SN:  50004  2.5666e-01 8.3385e-05 -6.7122e-01 -1.5304e-01
```

A full list of allowed SIMGEN_DUMP variables can be printed to the screen by specifying zero variables as follows:

```
snlc_sim.exe mysim.input SIMGEN_DUMP 0
```

After printing the variables, the program quits.

3.20.4 Rest-Frame Model Dump

```
GENRANGE_DMPTREST: -20  80  # dump rest-frame model for this Trest range
GENMAG_SMEAR:       0.0
GENMODEL_ERRSCALE:  0.0
```

3.21 Blind SN Samples

"Blind" SN samples can be generated such that there is no header information to indicate the SN type or how it was generated. This option can be used, for example, to blindly mix SN types (Ia, II, Ibc) for testing SN-classifiers. The blind-test option is invoked with

```
GEN_SNDATA_SIM:  25  # 1(verbose output) + 8(BLIND) + 16(RANDOM CID)
      or
GEN_SNDATA_SIM:  26  # 2(terse output) + 8(BLIND) + 16(RANDOM CID)
```

The 16-bit causes the integer CID to be randomly selected (1-900,000) so that mixing different SN samples cannot be sorted by CID. The 8-bit causes all of the SIM_XXX variables to be suppressed, along with any other information that could give hints about how the SN was generated. The 2-bit selects the terse output, which may be simpler for non-SNANA applications. You can also select the 1-bit (i.e., 25 or 27) to get the verbose output needed to analyze with SNANA. To generate the same "un-blinded" SN sample with the header info, set the "GEN_SNDATA_SIM" mask to 17,18, or 19 so that you get the same random CIDs.

If the host-galaxy simlib is selected (HOSTLIB_FILE keyword), then REDSHIFT_FINAL is set to the host-galaxy redshift and its error; otherwise REDSHIFT_FINAL is set to zero. The SIMGEN_DUMP option (§3.20.3) will store the correct values.

Note that the keyword CIDOFF plays the role of selecting a unique set of random CIDs so that merged samples will not have overlapping CIDs. For example, suppose you generate 1000 type Ia SNe with CIDOFF: 0. The CIDs will be the first 1000 randomly selected (and non-repeating) integers between 1 and 900,000. Now suppose you generate 1000 type II with CIDOFF: 1000. The simulation will generate 2000 CIDs, but only use the last 1000 on the list (i.e., skip the first 1000). When you combine the type Ia and type II SNe into one directory, the CIDs will not overlap, and the SN types (Ia and II) will be perfectly mixed with no correlation between type and CID. It is up to the user to pick the correct CIDOFF value for each SN type. After merging the SN samples, a useful unitarity check would be to do 'ls *.DAT | wc' and make sure that the total number of files matches the sum from the simulation jobs.

3.22 Forcing fixed Signal-to-Noise Ratio

Sim-input keyword

```
FUDGE_SNRMAX: 25
```

adjusts the exposure time for each event and filter so that the S/N ratio at peak brightness is fixed to the specified value.

3.23 Including a Second Sim-Input File

A sim-input file can be split into two files using the keyword

```
INPUT_FILE_INCLUDE:  my2nd.input
```

which instructs the simulation to read and parse "my2nd.input" in exactly the same way as the original sim-input file. To see why this might be useful, consider the non-Ia simulation that has many "NON1A:" keywords. The NON1A keys can be stripped out into a separate file such as NON1A_keys.input, and then included in many sim-input files. Thus a dozen sim-input files can each include NON1A_keys.input. To modify or add a NON1A key for all of the sim-input files, only one file needs to be modified.

3.24 Multi-dimensional GRID Option

Instead of generating random distributions in the variables describing each SN (redshift, luminosity parameter, color, etc ..), the simulation can generate SNe on a well-defined grid for each parameter using the following sim-input options,

```
GENSOURCE:       GRID    # replaces RANDOM option
NGRID_LOGZ:        20  # log10(redshift)
NGRID_LUMIPAR:     10  # x1, Delta, stretch,dm15 ...
NGRID_COLORPAR:     2  # AV or SALT2 color
NGRID_COLORLAW:     1  # RV or BETA
NGRID_TREST:       56  # rest-frame epoch
GRID_FORMAT:     FITS  # TEXT or FITS

GENRANGE_REDSHIFT:  0.01  1.2      # redshift range
GENRANGE_DELTA:   -0.4    1.8      # delta-range (mlcs only)
GENRANGE_RV:        2.2   2.2      # range of CCM89-RV
GENRANGE_AV:        0.0   2.00     # AV range
GENRANGE_TREST:   -20.0  90.0      # test epoch relative to peak (days)
GENFILTERS:       griz
```

and explicit examples of complete sim-input files are in

```
$SNDATA_ROOT/analysis/sample_input_files/GRID
```

This GRID option allows external (non-SNANA) fitting programs to use the SNANA models. The original motivation is for the "psnid" (photometric SN id) program used by the SDSS–II. Each NGRID_XXX value divides the corresponding GENRANGE_XXX range into the specified number of bins for the grid. The "GRID_FORMAT: TEXT" option produces a human-readable file intended only for visual inspection. The "GRID_FORMAT: FITS" option produces a platform-independent file to be read by external programs. To save memory for programs reading the FITS tables, the magnitudes and errors have been multiplied by 1000 and stored as 16-bit (2-byte) integers. Magnitudes dimmer than 32 are written as 32000, and undefined model magnitudes are stored as -9000 (i.e, mag$= -9$).

The GRID file is written in the same directory as the auxiliary files

```
$SNDATA_ROOT/SIM/MY_VERSION/MY_VERSION.GRID
```

where "GENVERSION: MY_VERSION" is specified in the sim-input file. Note that only the GRID file is written; no light curve files are written out.

The FITS tables can be visually examined using a utility such as "fdump" or "fv." SNANA has a "fits_read_SNGRID" utility to read in the generated GRID; to use this utility the following code-lines must be included,

```
#define SNGRIDREAD // use only the read utilities in sngridtools.c
#ifdef  SNGRIDREAD
#include "fitsio.h"
#include "sngridtools.h"
#include "sngridtools.c"  // fits_read_SNGRID is in here
#endif
```

The read-back utility fills the following global arrays/structures in sngridtools.h,

```
GRIDGEN_INFO
GRIDGEN_SURVEY  GRIDGEN_MODEL  GRIDGEN_FILTERS
PTR_GRIDGEN_LC  I2GRIDGEN_LCMAG I2GRIDGEN_LCERR
```

Also note that genmag_snoopy.c illustrated how to read and access the GRID.

Here is a brief description of the FITS tables and how to look up the correct magnitude and error from a set of SN parameters. Technically only the first (SNPAR-INFO) and last (I2LCMAG) tables are needed; the intermediate tables provide additional information that you would otherwise have to compute on your own. The SNPAR-INFO columns NBIN, VALMIN and VALMAX are simply copied from the sim-input parameters. The BINSIZE is calculated from the previous parameters, and the ILCOFF are used to determine the absolute light curve index (ILC) as a function of the SN parameters as follows:

$$\mathrm{ILC} = 1 + \sum_{i=1}^{4} \mathrm{ILCOFF}_i \times (\mathrm{INDX}_i - 1) \tag{7}$$

The parameter index $i = 1,4$ runs over (1) redshift, (2) luminosity parameter, (3) color (A_V or c), and (4) color law (R_V or β). Each integer index $\mathbf{INDX}_i$ runs from 1 to $\mathtt{NGRID}_i$ for parameter i. Do NOT extend the summation to include the filter and epoch indices. While the physical grid-values corresponding to each $\mathbf{INDX}_i$ can be computed from the SNPAR-INFO table, these grid values have been store in the intermediate tables that have a GRID suffix (and include FILTER-GRID and TREST-GRID).

Note that the $\mathbf{ILCOFF}_i$ are fixed, while the $\mathbf{INDX}_i$ depend on the set of SN parameters. For example, consider a redshift range of 0.01 to 1 and 200 bins; in logz space we have $-2 \leq \log_{10}(z) \leq 0$ and a logz binsize of 0.01. For $z = 0.1$, $\log_{10}(z) = -1$ and the GRID-index is $\mathbf{INDX}_1 = 100$.

Now we have an ILC index corresponding to a Supernova described by the four parameters above. Each SN light curve is written out in all of the GENFILTERS, and all of the SNe are strung together in the I2LCMAG table. This table contains one column of model magnitues and another column of model errors, each multiplied by 1000 to maintain millimag precision in 2-byte integer storage. The starting location in the I2LCMAG table is given in a separate 'pointer table' by PTR_I2LCMAG(ILC). Note that you could compute this pointer as

```
   PTR_I2LCMAG[ILC] = 1 + ((NGRID_FILT * NGRID_TREST) + NWDPAD) * (ILC-1);
```

where NGRID_FILT is the number of "GENFILTERS" and NWDPAD= 4 is the number of pad-words. Starting at the specified pointer location for ILC, the first word is a pad-word (-1111) and the second word is the first 8 bits of ILC; read-programs should verify these words to avoid getting lost. The next NGRID_TREST words are the magnitudes ($\times$ 1000) for the first filter (g), the next NGRID_TREST words are the magnitudes for the second filter (r), etc.. Finally, after reading all of the light curve magnitues there are two end-of-lightcuve pad-words with values of -9999.

The I2LCMAG storage for a single SN light curve is illustrated below:

```
# ----------------------------------------
   pad1  = -1111  <== start I2LCMAG address is PTR_I2LCMAG(ILC)
   pad2  =  first 8 bits of ILC
   I2LCMAG(g,ep1) = mag * 1000
   I2LCMAG(g,ep2)
   I2LCMAG(g,ep3)
   ...
   I2LCMAG(g,NGRID_TREST)
   I2LCMAG(r,ep1)
   ...
   I2LCMAG(r,NGRID_TREST)
   ...
   ...
   I2LCMAG(z,NGRID_TREST)
   pad3 = -9999
   pad4 = -9999
# ----------------------------------------
```

While pointers are provided to compute ILC and to determine the starting I2LCMAG address from ILC, you are on your own to find the sub-index corresponding to the filter and epoch.

For the non-Ia GRID, there is no physically meaningful luminosity parameter; this parameter is therefore used to store a sparse index that runs from 1 to the number of non-Ia templates that are specified with the "NON1A:" keyword in the sim-input file. The FITS file includes an additional NONIa-INFO table that gives the SNANA index, a character-string type (e.g., II, Ib, Ibc), and a character-string name of the underlying SN used to create the template (e.g., 'SDSS-002744').

4 The SNANA Fitter: snlc_fit

4.1 Getting Started Quickly

Here you will perform lightcurve fits, hopefully in under a minute. To get started,

```
> cp $SNDATA_ROOT/analysis/sample_input_files/MLCS/snfit_SDSS.nml .
    or
> cp $SNDATA_ROOT/analysis/sample_input_files/SALT2/snfit_SDSS.nml .

(Edit .nml file and put in correct VERSION_PHOTOMETRY = 'xxx')

> snlc_fit.exe  snfit_SDSS.nml >! snfit_SDSS.log &
```

When the command "ps" shows that the job has finished, congratulations ! You have fit your lightcurves with CERNLIB's MINUIT program.[5] If you are shaking your head wondering what the heck you just did, that's a good sign.

4.2 Discussion of Lightcurve Fits

Before reading this section, make sure you have successfully run the commands described in §4.1. Let's start the discussion by checking the end of the log-file,

```
>  tail snfit_SDSS.log
```

The very last line should be "ENDING PROGRAM GRACEFULLY." If you do not see this, check that your namelist variable VERSION_PHOTOMETRY is really pointing to an existing version in $SNDATA_ROOT/SIM. If you still have trouble, contact an expert for help.

Inside the input file snfit_SDSS.nml, the namelist variable

```
FITRES_DMPFILE = 'snfit_SDSS.fitres'
```

results in a dump of the fit parameters for each SN in a self-documented "fitres" file. Go ahead and "more snfit_SDSS.fitres" to see the results. The header keywords NVAR and VARNAMES specify the columns. The SNANA library includes a utility called RDFITRES to read these files. You can "cat" multiple fitres files together and add comments, and still read them with the same parsing code.

To figure out what you did, you need to check the namelist options in snfit_SDSS.nml. There are two separate namelists:

- &SNLCINP: defines selection of SNe and epochs by specifying criteria for the number of epochs, earliest & latest times relative to peak, maximum signal-to-noise, etc ... All namelist options are defined and commented inside SNANA_DIR/src/snana.car .

[5] http://wwwasdoc.web.cern.ch/wwwasdoc/minuit/minmain.html

- `&FITINP:` defines fitting options such as priors, marginalization, and range of $T_{\rm rest}$ in the second fit-iteration. All namelist options are defined and commented inside `SNANA_DIR/src/snlc_fit.car` .

In the sample namelist file, a prior on A_V is used by setting `PRIOR_AVEXP = 0.40`, which translates into a prior of the form $\exp(-A_V/0.4)$. There is no marginalization so that your first fits run much faster. To marginalize, set the number of integration bins per variable, `NGRID_PDF = 11`. Since the marginalization is over four fit variables (t_0, A_V, Δ, μ), the CPU-time goes as the fourth power of `NGRID_PDF`; previous studies indicate that 11 bins per fit-variable is a good compromise between accuracy and CPU time.

4.3 Methods of Fit-Parameter Estimation

There are three methods that can be used to estimate light curve fit-parameters:

1. **MINIMIZATION** based on CERNLIB's MINUIT program[6]. `&SNLC_INP` namelist parameter `NFIT_ITERATION` specifies the number of iterations (2 is recommended). You must always use this option, even if you use the options below.

2. **MARGINALIZATION** using multi-dimensional integration in `SNANA` function `MARG_DRIVER`. `&FITINP` namelist parameter `NGRID_PDF` controls the number of grid-points per fit-parameter (11 is recommended). You must run the minimizer first (`NFIT_ITERATION=2`) to get starting values and integration ranges. After marginalizing, the following crosschecks are performed: probability at the boundaries and number of bins with zero probability; if either is too large, the integration ranges are adjusted and the marginalization repeats.

3. **Monte Carlo Markov Chain (MCMC)**: See `&MCMCINP` namelist parameters.

[6] http://wwwasdoc.web.cern.ch/wwwasdoc/minuit/minmain.html

4.4 Initial Fit-Parameter Estimation

For MINUIT to converge, initial fit-parameters must be chosen so that the initial model light curve roughly overlaps the data. The color and shape parameter are fixed to an average value, and the distance modulus (or x_0 for SALT-II) is adjusted so that the model matches the data.

The estimate of the epoch of peak brightness (t_0) is usually read from the data file from the keyword SEARCH_PEAKMJD. If this keyword is missing, then SNANA will try to estimate t_0 by fitting a general function of the form (see SNANA function SET_PEAKMJD)

$$f(t) = A[1 + a_1(t-\bar{t}) + a_2(t-\bar{t})] \times \frac{\exp[-(t-\bar{t})/T_{\rm fall}]}{1+\exp[-(t-\bar{t})/T_{\rm rise}]} \ , \quad (8)$$

as suggested in the SNLS CC rate paper (Bazin et al, 2008). The fitted parameters for each filter are $A, a_1, a_2, \bar{t}, T_{\rm fall}, T_{\rm rise}$. The time at peak is obtained by setting the derivative equal to zero: $t_0 = \bar{t} + T_{\rm rise}\ln(T_{\rm fall}/T_{\rm rise} - 1)$. The initial $\bar{t}$ value is estimated to be the epoch with maximum flux. Each filter is fit independently and the final t_0 is the filter-weighted average. A filter's t_0 is dropped from the average if: (1) its max-flux measurement has the smallest S/N ratio among filters and has $S/N < 10$, or (2) its t_0 value is more than 30 days away from the average of the other filter-t_0 values (tested only if there are 3 or more filters). The max-flux epoch must have $S/N > 4$ to make an estimate of t_0.

Bit-mask options (lsb=1) via &SNLCINP namelist variable OPT_SETPKMJD are: bit-1) fix $a_1 = a_2 = 0$, bit-2) float a_1 and a_2, and bit-8) dump info for each SN. Explicit examples are :

```
OPT_SETPKMJD = 1    # fix a1 = a2 = 0 (default => no polynomial term)
OPT_SETPKMJD = 2    # float a1 & a2
OPT_SETPKMJD = 129  # fix a1=a2=0 and DUMP info for each SN
```

Users should compare the initial t_0 (from above) to the final t_0 from the light curve fit and check for outliers; outliers can be fixed by visual inspection of the light curve and setting the SEARCH_PEAKMJD keyword in the data file. To avoid clogging up the standard log-file, the MINUIT output for these fits is dumped to a separate log-file specified by namelist variable MNFIT_PKMJD_LOGFILE: the default filename is MNFIT_PKMJD.LOG.

WARNING (Jun 2010): The SN classifier challenge has uncovered two serious problems with the initial t_0 estimate for SNe Ia using Eq. 8. First, the filter-dependence of t_0 is not accounted for. Second, Eq. 8 can sometimes be very nonIa-like (even for a perfectly good fit), leading to t_0 estimates off by more than a week. Will need to add an option to use a more Ia-like function.

4.5 Fitting Priors

The fitting prior options in `snlc_fit.exe` are mainly designed to prevent catastrophic fits. There are also options related to the host-galaxy extinction. Photo-z priors are described in §4.8, and a brief description of the other `&FITINP` namelist prior options are given below.

- **PRIOR_MJDSIG**: sigma on Gaussian prior for MJD at peak brightness. Default is 20 days.
- **PRIOR_LUMIPAR_RANGE(2)**: range of flat prior for luminosity parameter. Typical values are {-0.5,2.0} for `MLCS2k2` parameter Δ, and {-5,+5} for SALT-II parameter x_1. Default range is {-9,+9}.
- **PRIOR_LUMIPAR_SIGMA**: sigma of Gaussian roll-off at edge of flat prior defined above. Default is 0.1.
- **PRIOR_DELTA_PROFILE(4)**: Used only for `MLCS2k2` Δ, the first two elements are $-\sigma$ and $+\sigma$ for the asymmetric Gaussian prior, the 3rd element is the Δ value at the Gaussian peak, and the 4th element is the minimum 'flat' probability for all Δ values within `PRIOR_LUMIPAR_RANGE`. A flat Δ prior is obtained by simply setting the 4th element to 1.0. Setting the 4th element to ~ 0.1 results in a Gaussian prior with a flat tail for large Δ values; this tail prevents the suppression of very fast decliners (91bg-like).
- **OPT_PRIOR**: Default is $1 \rightarrow$ use priors. Setting to zero turns off ALL priors regardless of their values.
- **OPT_PRIOR_AV**: Used only for `MLCS2k2` model, default is $1 \rightarrow$ use A_V priors. Setting to zero turns off A_V-related priors.
- **PRIOR_AVEXP(2)**: For `MLCS2k2` only, defines up to two exponential slopes for A_V prior.
- **PRIOR_AVWGT(2)**: For `MLCS2k2` only, defines weight for the two A_V-exponential terms.
- **PRIOR_AVRES**: Since the A_V prior has a sharp boundary at $A_V = 0$ and therefore a discontinuity in the fitting function, this `PRIOR_AVRES` option allows a Gaussian smearing of the prior function that results in a continuous function. Recommended values are .001 to 0.01.

4.6 Selecting an Efficiency Map for MLCS2k2 Prior

The MLCS2k2 prior includes a simulated efficiency as a function of redshift, Δ, extinction (A_V), and color law (R_V). There are two ways to select the efficiency map via the &FITINP namelist:

```
! pick default simeff file for your survey:
! $SNDATA_ROOT/models/simeff/simeff_[SURVEY].dat
  OPT_SIMEFF = 1
!
!    or
!
! select file name explicitly. Will first check YOUR
! current working directory; if not there, then fitter
! checks $SNDATA_ROOT/models/simeff/mysimeff.dat
  SIMEFF_FILE = 'mysimeff.dat'
```

The efficiency map is defined on a 4-dimensional grid (z, Δ, A_V, R_V), and interpolation is used to determine the efficiency for any values. The fitter automatically generates diagnostic plots with hid = 451-456. See §3.18 for a script to generate an efficiency map.

4.7 Viewing Lightcurve Fits

There is a PAW macro to look at the fits. If you have never done this before, then you need to run a once-in-a-lifetime command

```
>  paw_setup.cmd
```

Now go into paw and do

```
PAW > h/file 1 snlc_fit.his
PAW > snana#fitres
      or
PAW > snana#fitres tmin=-20 tmax=60
```

The dots are data, green curve is best-fit model, and blue curve is the generated lightcurve from the simulation.

If you really don't like using PAW, then you can run an after-burner script to prepare post-script plots of each lightcurve: try the command

```
> mkfitplots.cmd  --h snlc_fit.his
```

The "snlc_fit.his" argument above is the name of the histogram file that was specified with the namelist argument HFILE_OUT inside the fitter-namelist file. The mkfitplots script generates a lot of screen-dump that you should ignore. When finished, there should be two ps files,

```
snlc_fit_fits.ps
snlc_fit_marg.ps
```

The first file shows the *gri* lightcurve fits for each SN; the second file shows the probability distribution for each fit variable marginalized over the other three variables.

To view the light curves without doing any fits, set &SNLCINP namelist variable OPT_LCPLOT=1, run the snana.exe program, and then run the above PAW commands.

4.8 PhotoZ Fits

Here we describe light curve fits that determine the SN Ia redshift (z) from photometry, called "SN-photoZ" fits. There are two fundamental methods to perform photoZ fits. The first method, called a "constrained photoZ fit," is designed to identify SNe Ia that do not have a spectroscopic redshift: uses include SN rates and targeting host-galaxy redshifts for unconfirmed SN Ia. For `MLCS2k2` photoZ fits, there are four floated parameters: z, t_0, Δ, and A_V. For SALT-II photoZ fits, the four floated parameters are z, t_0, x_1, and c. The distance modulus is constrained (calculated) assuming a particular cosmology: $\mu = \mu(z, \Omega_M, \Omega_\Lambda, w)$ where z is floated in the fit, and $\Omega_M, \Omega_\Lambda, w$ are fixed by the user. The cosmology can be specified with &SNLCINP namelist parameters `H0_REF`, `OMAT_REF`, and `OLAM_REF`. For SALT-II photoZ fits, the distance modulus is converted into the x_0 parameter, and therefore `SALT2alpha` & `SALT2beta` must be specified as &FITINP namelist parameters.

The second method, called a "cosmology-photoZ" fit, involves floating five parameters: μ, z, t_0, Δ, and A_V for `MLCS2k2`, and x_0, z, t_0, x_1, and c for SALT-II. This method is designed to use large photometric samples to measure distance moduli that can be used to measure cosmological parameters. One of the difficulties with the 5-parameter fit is CPU time: the marginalization takes a few minutes per fit, so studying the bias on a sample of 10^4 simulated SNe Ia requires about a CPU-month of resources.

In addition to the two main methods above, there are variations that involve using the host-galaxy photoZ (host-photoZ) as a prior to help constrain the redshift. A distance-modulus prior can be applied to the second method (5-parameter fit); this is essentially a constrained-photoZ fit, but the photoZ errors will include uncertainties from the cosmological parameters. Needless to say, *never* run a cosmology fit on ouput where a distance-modulus prior is used !

There are five `&FITINP` namelist parameters that control photoZ fits. The default values are set so that photoZ fits are turned off,

```
DOFIT_PHOTOZ     = F
OPT_PHOTOZ       = 0      ! 1=>hostgal photZ prior; 2=> specZ prior
INISTP_DLMAG     = 0.1    ! 0=> constrain DLMAG; non-zero => float DLMAG
PRIOR_ZERRSCALE  = 100.0  ! scale error on host-photoZ prior
PRIOR_MUERRSCALE = 100.0  ! scale error on distance modulus prior
```

Setting `DOFIT_PHOTOZ=T` and `INISTP_DLMAG=0.0` results in a 4-parameter constrained-photoZ fit. Since `PRIOR_ZERRSCALE=100.0` by default, the host-photoZ errors are multiplied by 100 and therefore have no impact on the fits. Setting `PRIOR_ZERRSCALE = 1.0` results in using the host-photoZ prior described by a Gaussian distribution[7].

To switch from a constrained-photoZ fit to a 5-parameter cosmology-photoZ fit, simply set `INISTP_DLMAG` to any non-zero value such as 0.1. The use (or non-use) of the host-photoZ prior is controlled by the value of `PRIOR_ZERRSCALE`. The parameter `OPT_PHOTOZ` controls the source of the redshift prior. When you set `DOFIT_PHOTOZ=T`, `OPT_PHOTOZ` is automatically set to 1 so that the host-photoZ prior is used. If you set `OPT_PHOTOZ=2`, the spectroscopic redshift is used as a prior: this option is designed solely as a sanity check on the light curve fitter, and is not meant to use for science. If you set `OPT_PHOTOZ`> 0,

[7] Depending on user interest, non-Gaussian tails may be added later.

the DOFIT_PHOTOZ flag is automatically set, and vice-versa: thus you can turn on the photoZ option with either namelist variable.

If PRIOR_MUERRSCALE is 100 or larger (the default), then there is no prior applied to the distance modulus (μ). Setting PRIOR_MUERRSCALE = 2 will apply a μ-prior using a Gaussian profile of width $\sigma = 2\sigma_\mu$, where σ_μ is calculated from the user-specified uncertainties in the cosmological parameters. Note that OMAT_REF, and W0_REF are two-dimensional arrays that specify both the value and error. For example,

```
OMAT_REF =  0.3, 0.03
W0_REF   = -1.0, 0.1
```

would use $\sigma_w = 0.1$ and $\sigma_M = 0.03$ to calculate the μ-error for the photoZ at each fit-iteration.

To get going quickly, some useful examples of setting the namelist options are given below in Fig. 4.8.

A few other photoZ-related issues are:

- To test the photoZ methods in simulated SN Ia samples, the simulation includes an option to include host-galaxy photoZs based on an externally-supplied library (§ 3.14). To test SN-only photoZ fits (without host), increase GENSIGMA_REDSHIFT so that the initial redshift estimate is poor.

- To test the photoZ sensitivity to the initial redshift estimate, you can arbitrarily shift the redshifts using &SNLCINP namelist parameter REDSHIFT_FINAL_SHIFT.

Figure 2: Examples of setting photoZ options within the &FITINP namelist.

```
! =====================================================
! constrained photoZ fit, ignore host-galaxy photoZ:
  DOFIT_PHOTOZ    = T
  PRIOR_ZERRSCALE = 100.0  ! inflate error on photoZ prior
  INISTP_DLMAG    = 0.0    ! fix MU = MU(zphot,cosmology)

! equivalent way to do the above
  OPT_PHOTOZ      = 1
  PRIOR_ZERRSCALE = 100.0  ! inflate error on photoZ prior
  INISTP_DLMAG    = 0.0    ! fix MU = MU(zphot,cosmology)

! constrained photoZ fit using host-galaxy photoZ:
  DOFIT_PHOTOZ    = T
  PRIOR_ZERRSCALE = 1.0
  INISTP_DLMAG    = 0.0    ! fix MU = MU(zphot,cosmology)

! constrained photoZ fit, host-galaxy photoZ errors inflated by 1.3
  DOFIT_PHOTOZ    = T
  PRIOR_ZERRSCALE = 1.3
  INISTP_DLMAG    = 0.0    ! fix MU = MU(zphot,cosmology)

! cosmology photoZ fit, ignore host-galaxy photoZ:
  DOFIT_PHOTOZ    = T
  PRIOR_ZERRSCALE = 100.0  ! inflate error on photoZ prior
  INISTP_DLMAG    = 0.1    ! float DLMAG

! cosmology photoZ fit using host-galaxy photoZ:
  DOFIT_PHOTOZ    = T
  PRIOR_ZERRSCALE = 1.0
  INISTP_DLMAG    = 0.1    ! float DLMAG

! cosmology photoZ fit with priors on both distance & host-galaxy photoZ:
  DOFIT_PHOTOZ     = T
  PRIOR_ZERRSCALE  = 1.0   ! use host-galaxy photoZ errors
  PRIOR_MUERRSCALE = 3.0   ! use x3 mu-error calculated from dw & dOM
  INISTP_DLMAG     = 0.1   ! float DLMAG
! =====================================================
```

4.8.1 Redshift-Dependent Selection in PhotoZ Fits

There is a subtle fitting issue concerning the usable observer-frame filters for which $\lambda_{obs}/(1+z)$ is within the valid λ_{rest}-range of the light curve model, and for which $T_{rest} = T_{obs}/(1+z)$ are valid. In addition, requirements on quantities such as the min & max T_{rest} are ambiguous before the photoZ fit has finished, yet it is useful to make such cuts before fitting to prevent fitting pathological light curves and to reduce processing time. Here we discuss how to select filters and how to make cuts on T_{rest}-dependent quantities.

For regular cosmology fits using spectroscopic redshifts, a list of usable filters & the T_{rest}-range is made before the fit starts. For a photoZ fit, however, it is not clear which filters & epochs are valid until the fit has finished. For example, consider a *gri* photoZ fit using SALT-II: when $Z_{phot} < 0.072$, *i*-band maps to rest-frame wavelengths greater than the 7000 Å cutoff in SALT-II. Including *i*-band in the fit results in using an unphysical region of the model, while dropping *i*-band measurements results in a discontinuous drop in the χ^2. In the latter case, the minimizer is trapped in this low-χ^2 well, and quite often the minimum occurs at the drop-out boundary $Z_{phot} = 0.072$. Another example is in DES & LSST, where *g*-band maps below 3000 Å at redshifts above about 0.5.

To make initial T_{rest}-dependent cuts before the photoZ fit has started, the cuts are loosened by a factor of "$1+Z_{max}$", where $Z_{max} =$ `PHOTOZ_BOUND(2)` is the maximum allowed redshift (specified in &`FITINP`). The T_{rest}-cuts are therefore loosened to be valid for any redshift in the range specified by `PHOTOZ_BOUND`. For example, consider `PHOTOZ_BOUND = 0,1` and and a requirement that the min-T_{rest} is before -4 days; the initial cut would be a requirement of an epoch before -2 days using whatever `REDSHIFT_FINAL` is in the data file, and the -4 day requirement is applied after the photoZ fit has finished. Similarly, a max-T_{rest} requirement of 30-60 days is loosened to 15-120 days before the photoZ fit. If a filter is dropped after the first fit-iteration (see below), the loose T_{rest}-cuts are re-applied before fitting again.

To select observer-frame filters, the basic strategy is to perform the first-iteration photoZ fit with all filters except for those in the UV with $\lambda < 4000$ Å. A reasonable analytical extrapolation of the model beyond the defined wavelength range is required. This possibly biased photoZ is then used to determine which filters to exclude (or add in case of UV filter) in the next iteration. Technically, when one more more filters is excluded, the first-iteration is repeated so that two complete fit-iterations are performed with the correct filters. The basic assumption in this strategy is that it does not matter if there is an unknown bias in choosing the redshift to drop a filter ... as long as the fit, with or without the filter in question, is unbiased. As an example, consider photoZ fits with *griz* filters. For $z > 0.49$, *g*-band should be excluded. As a safety margin, one might exclude *g*-band when the 1st-iteration photoZ value is above 0.43, or 0.44, or 0.45 ... the cut does not matter as long as we are confident that when *g*-band is used, it is within the valid range of the model.

The redshift safety margin is controlled by namelist parameters

```
PHOTOZ_ITER1_LAMRANGE = 4000, 25000 ! 1st-iter obs-frame lambda range
PHOTOZ_BOUND          = 1.0E06, 1.4 ! hard MINUIT bound
PHOTODZ_REJECT        =  0.05  ! dz cut
PHOTODZ1Z_REJECT      = -99.   ! dz/(1+z) cut; default is no cut
```

The default `PHOTOZ_ITER1_LAMRANGE` cut is set to exclude any UV filter on the first iteration since this

filter is used only at the lowest redshifts. Note that the excluded UV filter can be added back after the first fit-iteration if the photoZ value is low enough. PHOTOZ_BOUND is a hard MINUIT bound to prevent crazy excursions during the minimization, and is also used to loosen the $T_{\rm rest}$-related cuts before the fit has started.

The next two cut-parameters define the redshift safety margin, and can be defined as a cut on *dz* and/or $dz/(1+z)$. The SNANA default is to use only the cut on *dz*, so we use this for discussion, noting that the other cut works in a similar manner. In principle it would be better to cut on the number of fitted σ_z, but on the first iteration the MINUIT errors are sometimes pathological; it is therefore safer to make a fixed cut.

The *dz* cut is illustrated here using *g*-band and the MLCS2k2 model for which the valid wavelength range is 3200-9500 Å. Since the mean filter wavelength is $\lambda_g = 4790$ Å, the valid rest-frame redshifts are given by $Z_{\rm min} = \lambda_g/9500 - 1 = -0.50$ and $Z_{\rm max} = \lambda_g/3200 - 1 = +0.50$. The *g*-band is excluded if the 1st-iteration photoZ satisfies $Z_{\rm phot} > Z_{\rm max} -$ PHOTODZ_REJECT; in this example, $Z_{\rm phot} > 0.45$. For *Y*-band ($\lambda_Y = 10095$ Å), $Z_{\rm min} = 0.062$ and this filter is excluded if $Z_{\rm phot} < Z_{\rm min} +$ PHOTODZ_REJECT, or $Z_{\rm phot} < 0.11$. Note that PHOTODZ_REJECT is defined to be positive when adding a safety margin, regardless of whether a blue or red band is being tested. Setting PHOTODZ_REJECT to a large negative value (i.e, -99) disables the cut. Finally, the PHOTODZ_REJECT cut is applied to all observer-frame filters, and often more than one filter (i.e, *ugr*) is rejected.

A quick list of dropped filters can be viewed from the log-file with the following 'grep' command:

```
grep "DROPPED" myjob.log
      WARNING for SN 40002 : DROPPED obs-filter=g
      WARNING for SN 40002 : DROPPED obs-filter=r
```

and similarly grepping for "ADDED" will find any (UV) filters that were added.

To study filter dropouts in more detail, five variables are include in the fitres-ntuple (ntid 7788):

- NEARDROP: index of filter that is closest to being dropped. Positive (negative) value indicates that this filter was included (excluded) after the first fit-iteration.
- DZMIN1: redshift safety margin for filter NEARDROP after 1st iteration. Positive value always indicates that it is within the valid region of the model; negative value indicates invalid region.
- DZMIN2: same as above, but after 2nd fit-iteration.
- DZ1ZMIN1 & DZ1ZMIN2: same as above, but for $dz/(1+z)$.

TRAINING & TUNING CUTS:

SNe with spectroscopic redshifts ($Z_{\rm spec}$) should be use to train the filter-selection cuts. For each filter labeled by index "IFILT," plot $Z_{\rm spec}$ when NEARDROP = IFILT and check that there are no (or very few) entries in the undefined redshift-region of the model. Also plot $Z_{\rm spec}$ when NEARDROP = -IFILT, and you will see entries in the undefined region, but this is OK since the filter was dropped. $Z_{\rm spec}$ values in the defined region indicates that this dropped filter could have been safely used, but was rejected based on its photoZ value from the first fit-iteration. Users will have to decide the trade off between including a particular filter more often in the defined region versus using that filter more often in the undefined region.

4.8.2 Smooth Model Error Transition Across Filter Boundaries

For rest-frame models such as MLCS2k2, the model error changes abruptly when a photoZ variation results in an observer-frame filter mapping into a different rest-frame filter. This abrupt change in the model error causes a small kink in the χ^2, and can cause fitting pathologies. This pathology_is treated by smoothly transitioning the model error between $\pm$200 Å of the transition wavelength (λ). The 200 Å range can be modified with the namelist parameter LAMREST_MODEL_SMOOTH. The transition weight-function is an arc-tangent that is scaled to equal 0 at $\lambda - 200$ and 1 at $\lambda + 200$. (see function RESTFILT_WGT for more info).

4.8.3 Don't Fool Yourself when PhotoZ-Fitting Simulations

When studying photoZ fits with simulations, avoid fooling yourself with simulations outside the valid wavelength range. If you use the default light curve model in the simulation, measurements outside the valid wavelength range are excluded, and therefore the fitter has the same "initialization" advantage in picking filters as using a spectroscopic redshift. For testing photoZ fits, a "wavelength-extended" model should be used as explained in §4.11. Even with a wavelength-extended model, you can fool yourself because the same model is used in both the simulation and in the fit; hence even if the extrapolated model (i.e, below 3200 and above 9500 Å for MLCS2k2) is wrong in reality, it is by definition correct when fitting the simulation. This forced correctness of the extrapolated model means that the 1st fit-iteration using all of the filters is guaranteed to give good results. A better test is to fit with a light curve model that deviates from the simulated model in the extrapolated regions. The 1st fit-iteration should then produce biased photoZ estimates, and ideally the filter-exclusion cut will work well enough so that there is no bias after the 2nd fit-iteration.

4.9 Optional Redshift Sources

Some surveys may include more than one redshift source such as spectroscopic redshifts from an external collaborator (e.g., BOSS redshifts for SDSS targets), or photometric redshifts obtained from different methods. While the "REDSHIFT_FINAL" key specifies the nominal redshift, optional redshifts and associated typings can be used with the namelist variable

```
&SNLCINP
   ZEXTRA_SOURCE = 'ZZZ'
   ...
&END
```

where "ZZZ" is the name of the redshift source. The SNANA programs will then search each SN data file for the optional keywords

```
[ZZZ]_TYPE: nnn                                      # optional
[ZZZ]_REDSHIFT_HELIO:  0.04592 +- 0.0002  (Helio)    # optional
[ZZZ]_REDSHIFT_CMB:    0.04500 +- 0.0002  (CMB)      # optional
[ZZZ]_END:                                           # mandatory key
```

where "nnn" is the SN type based on using the optional redshift. The first three keys are optional, meaning that they only need to be specified in data files where such information exists. The "[ZZZ]_END:" key is required in every data file; if this key is missing, the program will abort. Several different sets of optional redshifts can exist in each data file, but they must all go at the end of the header (just before the first observation).

These optional redshifts may be used only by a list of valid users specified in a global file called

```
more  $SNDATA_ROOT/[SURVEY]/[ZZZ]_USERS.LIST
USER: <user1>
USER: <user2>
USER: <user3>
etc  ...
```

This file-system is not a security system, but is only intended to prevent accidental mis-use.

4.10 Excluding/Downweighting Filters and Epoch Ranges

Here we discuss options to exclude or down-weight filters and/or epoch ranges for the light curve fitter. The filter selection is based on rest-frame wavelength so that a uniform cut can be applied to different filter systems. To globally exclude U-band, for example, set `$SNLCINP` namelist variable

```
CUTWIN_RESTLAM = 3900.  20000.
```

which excludes filter(s) whose central wavelength satisfies $\bar{\lambda}_{\rm obs}/(1+z) < 3900$ Å. This option excludes the U-band data as if it were not part of the data file; hence U-band is not used for spectral warping, $T_{\rm rest}$ cuts, etc ...

One problem with the above option is that there is no way to extrapolate the model back to U-band and check data-model fit residuals. To keep track of data-model residuals for the excluded region, it is better to simply down-weight rather than exclude a particular range in wavelength or epoch. A set of four `FUDGE_FITERR_XXX` variables allow the user to down-weight arbitrary wavelength and epoch ranges. These four `$FITINP` namelist variables are illustrated in the following example:

```
FUDGE_FITERR_TREST(4)    = -20., 100.     ! rest-frame range (days)
FUDGE_FITERR_RESTLAM(2)  = 1000., .3900.  ! wavelength range (A)
FUDGE_FITERR_PASSBANDS   = 'ugriz'        ! observer filters
FUDGE_FITERR_MAXFRAC     = 10.            ! error -> 10*maxFlux
```

This example does essentially the same thing as the above example using "`CUTWIN_RESTLAM = 3900., 20000.`" However, using the `FUDGE_FITERR_XXX` variables means that filters mapping onto rest-frame U-band are used in the spectral warping for K-corrections, and these filters are also used for the $T_{\rm rest}$ cuts. In the fit, an additional uncertainty of $10\times$ the maximum flux (`FUDGE_FITERR_MAXXFRAC=10`) is assigned to each epoch that satisfies the `RESTLAM` and `TREST` windows, and hence such epochs are effectively excluded from the fit. Note that `FUDGE_FITERR_PASSBANDS` specifies the observer-frame filters for which the other criteria apply. If you set the observer passbands to 'u', then the `RESTLAM` and `TREST` criteria are applied only for observer-u and not the other filters.

Here is another example in which epochs before -5 days and after $+50$ days are excluded for all filters:

```
FUDGE_FITERR_TREST(4)    = -99, -5.0,  +50., 999.
FUDGE_FITERR_RESTLAM(2)  = 1000., 20000.
FUDGE_FITERR_PASSBANDS   = 'ugriz'
FUDGE_FITERR_MAXFRAC     = 10.
```

4.11 Rest-Frame Wavelength Range

Each SN model includes a function that returns the valid rest-frame wavelength range (λ_{rest}-range) to the fitting program. There are two different ways to change the λ_{rest}-range, but note that you can only make the λ_{rest}-range *more restrictive*; i.e, you cannot arbitrarily loosen the λ_{rest}-range for a SN model. The two equivalent namelist options to change the λ_{rest}-range (Å) are

```
&SNLCINP
  CUTWIN_RESTLAM = 2000 ,  25000
&END

&FITINP
   RESTLAMBDA_FITRANGE = 3500 ,  9500
&END
```

The first option rejects measurements as part of the pre-fitting selection, and is useful if you want to apply this requirement without running the fit; i.e., using only the `snana.exe` program. The second option is applied during the fitting stage.

The λ_{rest}-ranges appear in your log-file as follows,

```
CUTWIN_RESTLAM =  2000.  25000.
SN-MODEL  LAMBDA RANGE:  3200. -  9500.
USER-FIT  LAMBDA RANGE:  3500. -  9500.
```

and again note that the *most restrictive* range is used. If you specify a wide open range such as 1000 to 30000, this simply tells the fitting program to use the default range.

Finally, if you really insist on changing the default λ_{rest}-range for a particular SN model,[8] you can modify the default range by editing the file

```
$SNDATA_ROOT/models/[model-subdir]/RESTLAMBDA_RANGE.DAT
```

Such changes should be used with great caution because this change affects all users. Before making such a change, consider creating a private model-version (i.e., `mlcs2k2.LAM2800`).

[8] All maintenance warranties are null & void if you change the default λ_{rest}-range.

4.12 Extracting Light Curve Shape from the Fit

The light curve shape, defined as the flux-to-peakFlux ratio (F/F_0) versus epoch, is determined after each light curve fit. In principle, this ratio is unity at the epoch of peak brightness. The residual-ntuple variable is FPKRAT (ntuple id 7799) and the fortran variables are EP_FPKRAT and EP_FPKRATERR. See fortran subroutine FPKRAT for example on how to access these arrays.

While the flux values are from the data, the estimate of the peakFlux is based on the best-fit model. The peakFlux estimate can be significantly off, particularly for multi-band light curves that fit one of the colors poorly. Such peakFlux errors are evident for light curves in which the the data points in a given filter lie well above or below the best-fit model. To get a better estimate of the peakFlux, one can correct for the average data/model ratio in a particular epoch-range. This correction is implemented using the $FITINP namelist variable

```
TREST_PEAKRENORM = -10.0,  +20.  # rest-frame days
```

which specifies the rest-frame range for which to include epochs in the data/model correction. The default values are 0,0 $\rightarrow$ no correction. A data/model weighted-average is taken over the epochs within TREST_PEAKRENORM. The weight (w_i) at each epoch "i" is defined to be

$$w_i \equiv \frac{1}{\sigma_i^2 \times (|T_{\rm rest}| + 1)} \ , \tag{9}$$

where σ_i is the data/model flux-ratio uncertainty based on the data-flux error (i.e., ignores the model error), and "$|T_{\rm rest}| + 1$" is an arbitrary factor (in days) used to downweight epochs away from peak.

To see the peakFlux estimates on the light curve fit (§ 4.7), use the command

```
mkfitplots.cmd  --h <hisfile>  PEAKFLUX
   or
PAW > snana#fitres pkf=1
```

and a pink horizontal line is drawn through the peakFlux estimate for each filter and SN. The user is encouraged to vary TREST_PEAKRENORM to check the sensitivity of the epoch range.

4.13 Landolt ↔ Bessell Color Transformations

As explained in the first-season SDSS–II results paper (Appendix B of arXiv:0908.4274), the nearby SNe Ia magnitudes are reported in the Landolt system, but there are no *UBVRI* filter responses for this system. We therefore define color transformations between the Landolt system and synthetic *UBVRI* magnitudes using Bessell (1990) filter response functions (See Eqs. B1-B2 in above reference). These color transformations can be implemented in the fitter with the &FITINP namelist flag

```
OPT_LANDOLT = 1  ! transform rest-frame Landolt model mags -> Bessell90
OPT_LANDOLT = 2  ! transform obs-frame Bessell90 mags -> Landolt
OPT_LANDOLT = 3  ! both of the above
```

For example, to analyze the JRK07 or CFA3 samples with MLCS2k2, set OPT_LANDOLT=3; to analyze with SALT-II, set OPT_LANDOLT=2. To analyze ESSENCE data with MLCS2k2, OPT_LANDOLT=1. In the last case, the ESSENCE *RI* filter response functions are well known, so there is no need to use Bessell90 filter responses.

All of the above use BD17 as the primary reference. To use Vega, add 4 (3rd bit) to each OPT_LANDOLT value. You are also responsible for using the appropriate K-correction file with the matching primary reference.

4.14 Interpolating Fluxes and Magnitudes

There are two methods for interpolating the SN flux at a particular observation time (MJD). The first method is to use an SN Ia light curve model, and the second method is to use the generic 'any' function from §4.4.

For the first method, prepare a two-column file that lists each SNID and MJD to interpolate, and then specify the following &SNLCINP namelist variables,

```
SNMJD_LIST_FILE = 'KECK-SDSS.LIST'
SNMJD_OUT_FILE  = 'KECK-SDSS.OUT'
```

The interpolated fluxes and errors are dumped into a human-readable output file that has keywords so that you can easily extract needed information with a parsing routine. For each SN specified there are two interpolation entries in the output file: first is at the requested MJD, and second is at the epoch of peak brightness. If you only want the interpolated flux at peak brightness, specify MJD = 0 in the SNMJD_LIST_FILE.

Since the interpolation is based on the best-fit model, fitting a multi-color light curve can result in sub-optimal interpolations when a particular passband is not well fit. To get the most accurate interpolations, it is recommended to fit a single passband by either specifying one passband with the &FITINP namelist variable FILTLIST_FIT, or by downweighting the other passbands using the FUDGE_FITERR_XXX options (§4.10). Thus, to interpolate the flux in each of the *griz* passbands, four separate fits should be done. The &FITINP namelist parameters to interpolate *g*-band by downweighting the other bands are as follows:

```
FILTLIST_FIT = 'griz'
FUDGE_FITERR_TREST     = -20. 80. 0. 0.
FUDGE_FITERR_PASSBANDS = 'riz'
FUDGE_FITERR_MAXFRAC = 10.
```

Note that you can use command-line arguments (§7.1) to write a wrapper that loops over the filters,

```
snlc_fit.exe  myfit.nml  FUDGE_FITERR_PASSBANDS riz SNMJD_OUT_FILE  g.OUT
snlc_fit.exe  myfit.nml  FUDGE_FITERR_PASSBANDS giz SNMJD_OUT_FILE  r.OUT
snlc_fit.exe  myfit.nml  FUDGE_FITERR_PASSBANDS grz SNMJD_OUT_FILE  i.OUT
snlc_fit.exe  myfit.nml  FUDGE_FITERR_PASSBANDS gri SNMJD_OUT_FILE  z.OUT
```

If the SN Ia models do not give a decent fit to a particular light curve, which may happen in an ultraviolet band or for a peculiar Ia, then you may want to try the second method. This method is based on the "any-LC function" option designed to estimate the peak-MJD (§4.4). Note that you must run snana.exe instead of snlc_sim.exe! The &SNLCINP namelist parameters are

```
OPT_SETPKMJD = 1  # fit with any-LC function
OPT_LCPLOT   = 1  # make plots for PAW > snana#fitres .
```

Since each passband is fit independently, there is no need to mask off the other passbands.

The interpolation option above (SNMJD_LIST_FILE) works with the OPT_SETPKMJD option, but the results are printed to the screen instead of to a separate output file. This function (Eq. 8) can also be easily reconstructed from the fitted parameters that are dumped to MNFIT_PKMJD.LOG.

4.15 Selecting Telescope, Field and SNe

Here are example namelist options to select a telescope, field and SNe to process:

```
SNTEL_LIST    = 'SDSS'          ! list of telescopes
SNFIELD_LIST  = '82S' , '82N'   ! list of fields to process
CUTWIN_NFIELD = 1, 99           ! number of overlapping fields

CUTWIN_CID    =   700, 2000       ! Cand-ID range to process
SNCID_LIST    =   5944,   10550   ! list of SN to process
SNCCID_LIST   =  '5944', '10550'  ! same as above

SNCID_IGNORE  =   4524,   8151,   7017   ! list of SN to ignore
SNCCID_IGNORE =  '4524', '8151', '7017'  ! same as above
```

The list of valid telescopes and fields is in $SNDATA_ROOT/SURVEY.DEF, and you can select multiple telescopes and fields as illustrated above with SNFIELD_LIST. If you do not specify SNTEL_LIST or SNFIELD_LIST, then all telescopes or fields are processed by default; therefore, use these options only if you want to select a subset. If you select an invalid telescope or field, the code aborts. CUTWIN_NFIELD selects the number of overlapping fields. For example, to select SDSS SNe that overlap both 82N & 82S, set CUTWIN_NFIELD = 2, 2; i.e., require two overlapping fields.

The principle method for selecting SNe is the namelist variable CUTWIN_CID (CID = Candidate ID). If the SNe are identified by integers, as in the SDSS-II survey, then CUTWIN_CID simply selects the CID range. If the SNe are identified by character strings, then they are given internal CID values 1,2, ... up to the number of SNe. If you have 100 SNe defined as character strings, then set CUTWIN_CID = 1, 100. You can always open this window (1, 100000) to ensure that all SNe are processed. In addition to the CID range, you can specify a specific list of SNe to process (SNCID_LIST and/or SNCCID_LIST) and a specific list of SNe to ignore (SNCID_IGNORE and/or SNCCID_IGNORE). Both of these lists can be specified as integer or string. If SNe are specified with CUTWIN_CID and a list, then all specified SNe are processed; i.e., the logical-AND of all SN-selection.

In the "_LIST" variables, a zero or blank acts as a terminator. For example, SNCID_LIST = 5944, 0, 1032, 10550 would process SN 5944 and ignore the rest.

4.16 Creating Your Private Fitter: "snlc_fit_private.exe"

Here we describe a simple way to add your own fortran code into the fitter, such as writing out specific information to a text file, calculating a quantity that is not available, or testing modifications to the public code. There are two steps to creating your own private executable:

```
> cp $SNANA_DIR/src/snlc_fit_private.cra .
> snmake snlc_fit_private
```

should result in a private executable file in your directory: snlc_fit_private.exe. Now run your private version,

```
./snlc_fit_private.exe  myinput.nml
```

and the sample code will dump information for the first 4 SN. You can edit your private version of snlc_fit_private.cra and modify USRINI, USRANA and USREND. The sample USRANA shows how to access fit results, and how to access information for each epoch used in the fit. You can also re-name the code to any other name, such as 'myfit.cra', and then compile an executable with the command "snmake myfit" to produce the executable file myfit.exe.

In addition to adding private code into the USR[INI,ANA,END] routines, you can also modify the official public code. Copy any subroutine from $SNANA_DIR/src/snana.car or snlc_fit.car, paste it into your private snlc_fit_private.cra, and then make modifications. Once these changes are fully tested, you can request that the modified routine(s) be installed into the next public release of SNANA. Your changes may be included as the default, or they may be available to other users via input namelist flags. Note that modifications can also be made by checking out the entire SNANA product from CVS. You are welcome to check code out of CVS (cvs co), but please do NOT check code in without contacting the SNANA manager. Working with snlc_sim_private.cra should be much easier than working with the entire SNANA product.

WARNINGS:

- Do not trap yourself into an obsolete version of SNANA if you have lots of private code that is not integrated into the public SNANA.
- If you find yourself doing lots of cutting & pasting as part of your analysis, this is a bad sign: ask for help!
- If you find that you are doing lots of tedious/redundant operations, ask for help. Often adding just a few lines code into SNANA can greatly simplify your analysis procedure.

4.17 Private Data Path: PRIVATE_DATA_PATH

After creating or translating a new version of light curves, it is useful to run tests before copying these files to the public/official area $SNDATA_ROOT/lcmerge. SNANA and fitter jobs can read data from a private directory as follows:

```
&SNLCINP
   VERSION_PHOTOMETRY = 'myVersion'
   PRIVATE_DATA_PATH  = 'myDir'
   .
   .
&END
```

The version "myVersion" will be read from "myDir" in exactly the same way that it would have been read from $SNDATA_ROOT/lcmerge. Users are cautioned to use this feature only for testing, and not as long-term data storage for your analysis.

4.18 Mag-Shifts in Zero Points and Primary Reference Star

The snlc_fit program provides an interface to modify zero-point offsets as well as the magnitudes of the primary reference star. The primary magnitudes can be adjusted using the &SNLCINP namelist option

```
MAGOBS_SHIFT_PRIMARY  = 'B .02  V .02'
MAGREST_SHIFT_PRIMARY = 'B .02  V .02'
```

which shifts the primary mags by 0.02 for B and V in the example above. Note that separate strings are used for the observer-frame and rest-frame to allow for the same filter-character to be used in each reference frame. This option is equivalent to re-generating the K-correction tables with these shifts, but the MAG[OBS,REST]_SHIFT_PRIMARY option is much quicker since you can use the same K-correction tables.

For the zero-points, there are two ways to introduce shifts. The first method is to specify the offsets in a file, ZPOFF.DAT, located in the filters sub-directory. For the SDSS AB-offsets, the file location is

```
> more  $SNDATA_ROOT/filters/SDSS/ZPOFF.DAT
u -0.037  g 0.024  r  0.005  i  0.018  z  0.016
```

If ZPOFF.DAT does not exist, the shifts are zero. This method is used for standardized shifts.

The second option is designed to probe the uncertainty in the zero-point offsets; an arbitrary shift can be specified with the &SNLCINP namelist string

```
MAGOBS_SHIFT_ZP = 'g .02  r .02  i .02'
```

Note that the shifts in ZPOFF.DAT and MAGOBS_SHIFT_ZP are added so that you make well-defined shifts relative to the standard shifts in ZPOFF.DAT.

4.19 Updating the Filter Transmission for each SN

In 2009 the SNLS reported that their filter transmission depends on the focal plane position.[9] While their transmission function depends only on the radius from the center, in general the SN filter transmission can be unique for each SN. Using the SNANA fitting program, an SN-dependent filter transmission can be specified, although it is assumed that each transmission function is the same for all epochs. This feature is invoked by specifying an 'update' directory from the $SNLCINP namelist as follows:

```
FILTER_UPDATE_PATH = 'MYPATH'
```

MYPATH can be a subdirectory under $SNDATA_ROOT/filters, or MYPATH can be the full directory name; both directories are checked, with the former having priority. This directory must contain an instruction file called 'FilterFile.INFO', along with the filter transmission for each SN and filter specified in a separate text file. Rather than giving an explicit list of filter-transmission filenames, the instruction file provides a prefix and suffix used to construct the filenames. Using the following example for FilterFile.INFO,

```
PREFIX:  SNLS3year_
SUFFIX:  .dat
```

the filename for a given SNID and FILTER is "SNLS3year_[SNID]_[FILTER].dat."

WARNING: currently this feature works only for the SALT2 model; a future SNANA version will work for rest-frame models that use K-corrections.

[9] Regnault et al., **A&A 506** , 999 (2009).

4.20 Analysis Ntuples

Analysis variables and fit-parameters are reported in several different CERNLIB ntuples set by the following namelist flags:

```
&SNLCINP
  LTUP_SNANA = T        ! 7100: snana analysis variables
  LTUP_SKY   = F        ! 7101: moon, PSF, noise vs. filter and epoch
  LTUP_PHOTOMETRY = F   ! 7200: photometry for all epochs & filters
&END

&FITINP
  LTUP_FITRES     = T   ! 7788: analysis variables + fit-parameters
  LTUP_RESIDUAL   = T   ! 7799: data-model residuals vs. epoch & filter
&END
```

The default values (T or F) and ntuple id-numbers are given above. The SNANA ntuple (7100) is limited, but is useful to quickly study selection criteria without running fits (i.e., setting NFIT_ITERATION = 0 in the &SNLCINP namelist). Ntuple 7788 is the main analysis-ntuple with more than 100 variables, and ntuple 7799 is used to study light curve residuals, and to correlate flux-residuals with epoch, passband, PSF, sky-noise, photometry flag, etc ... To extract ntuple variable into a text file, see §7.4.

4.21 Analysis Alternative for those with HBOOK-phobia

The snana variables and fitter results are packed in 'HBOOK' histograms and ntuples. At the end of each job, these histograms and ntuples are written into a single HBOOK file, more commonly known as a "histogram" file (hence the common extension, ".his"). The utilities to create and access histogram files are part of CERNLIB. Fortran routines are available to access histogram contents from a compiled program, and PAW[10] is a well-known interactive program to analyze the contents of an HBOOK file, and to make plots. If you have little or no knowledge of HBOOK and PAW, and you are terrified at the prospect of being forced to learn these ancient (i.e., 20'th century) analysis tools, this section explains some simple alternatives on how to analyze the fitter outputs by simply translating the ntuple contents into column-formatted text files.

First, to see each light curve, best-fit model, and fit-parameters, run this script,

```
> mkfitplots.cmd  --h  snlc_fit.his
```

which creates one postscript file showing the light curve fit and results for each SN, and another postscript file showing the marginalized distributions for each fit-parameter.

Analysis variables are stored in ntuples (§4.20) which can be dumped into text files as explained in §7.4.

[10] PAW = Physics Analysis Workstation

4.22 Monitoring Fit-Jobs with "grep"

If you pipe the fitter screen dump to a log file, the unix "`grep`" command can be used to quickly monitor progress during the fits, as well as after the fits have completed. This is particularly useful when many fit-jobs are run in parallel so that you can monitor progress and catch aborts. Many screen outputs are explicitly designed to help monitor the job status. Below are some examples of useful grep-commands to run. A dagger (†) indicates those commands that work only when the fit-job has finished. Note that adding "`| wc`" to the end of a grep command will count the number of entries.

- `grep GRACE fit*.log` †
  lists the unique string `ENDING PROGRAM GRACEFULLY` to identify and count jobs that have finished.

- `grep " ABORT " fit*.log` †
  identifies jobs that have aborted. Note the blank space before and after `ABORT` to distinguish from namelist variables that include `ABORT` as part of the name.

- `grep "after snana" fit*.log` †
  shows the number of SN before and after `SNANA` cuts.

- `grep "after fit" fit*.log` †
  shows the number of SN after fitter cuts.

- `grep "PROCESS TIME" fit*.log` †
  shows total processing time.

- `grep "PASSES FIT" fit*.log`
  lists each SN that passes fit cuts.

- `grep "Cuts REJECT" fit*.log`
  lists each SN rejected by SNANA cuts.

- `grep "FAILED FIT" fit*.log`
  lists each SN rejected by fitter cuts.

- `grep ":DLMAG" fit*.log | grep pdf`
  lists the marginalized luminosity distance for each fitted light curve. Works for any fit-parameter: `AV, DELTA, PEAKKMJD, PHOTOZ`. Don't forget to include the colon, or you will be overwhelmed by the screen dump.

- `grep ":DLMAG" fit*.log | grep fitpar`
  lists the minimized luminosity distance for each fitted light curve.

- `grep MARGINALIZATION fit*.log | grep TOTAL`
  `grep MARGINALIZATION fit*.log | grep PRIOR`
  `grep MARGINALIZATION fit*.log | grep DATA`
  returns marginalized χ^2, N_{dof} and fit-probabilities for each SN.

- `grep "MINUIT MIN" fit*.log | grep TOTAL`
  `grep "MINUIT MIN" fit*.log | grep PRIOR`
  `grep "MINUIT MIN" fit*.log | grep DATA`
  returns `MINUIT`-minimized χ^2, N_{dof} and fit-probabilities for each SN.

5 Adding a New Survey

Starting with SNANA v6_00 you can add a new survey without modifications to the software. Primary SEDs, primary magnitudes and filter transmissions are defined via K-correction files, even for models like SALT-II that do not use K-corrections (but still use primary SEDs and magnitudes). A filter is defined by a single character to simplify the handling of a wide variety of output variable names that append the filter string (i.e, MAGT0_[filter]), and to simplify output formatting. While the SNANA filter definition is limited to the 1-character strings above, arbitrary filenames can be used to define the filter transmissions. There are 62 allowed filter-characters: A-Z, a-z, and 0-9. SNANA versions prior to v9_00 allowed only the legacy filters *ugriz*, *UBVRI*, *YJHK*, along with 0-9. Additional filter-characters will be allowed when we all switch to using Chinese keyboards. Here are the steps for adding a new survey:

1. Add your survey and telescope to the file $SNDATA_ROOT/SURVEY.DEF. Check if your survey and/or telescope is already defined before making changes.

2. Add new filters in $SNDATA_ROOT/filters. The wavelength spacings for the filter transmissions must be uniform. If the wavelength spacings are large (several hundred Å) you should prepare a finer-binned filter set using an appropriate interpolation algorithm.

3. Generate K-correction tables using kcor.exe,[11] and see §5.1 for rules about filter names. You can leave your private K-correction table(s) in your directory where you run other jobs, or you can share official K-correction tables in $SNDATA_ROOT/kcor/. For initial testing, use a very course grid so that the tables are built quickly. Once the simulation and fitter are working, you can generate the K-correction tables with a finer grid. The grid is controlled by REDSHIFT_BINSIZE and AV_BINSIZE. WARNING: you must generate a K-correction file even if you plan to run an observer-frame fitter such as SALT-II that does not use K-corrections; in this case do "kcor.exe mykcor.input SKIPKCOR" so that the K-corrections are skipped, but the filters and primary SED are included. Finally, for rest-frame models that use K-corrections, there is a limit of 10 rest-frame filters and 10 observer-frame filters. For observer-frame models such as SALT-II, the limit is 50 filters.

4. If you want to generate simulated samples, you need to prepare a simulation library. See examples in $SNDATA_ROOT/simlib. This is usually the most difficult part of setting up a new survey.

5. Check for an adequate rest-frame model to describe the supernova light curve.

6. If you plan to add new data files, use existing an data version as a template. To see existing versions do "cd $SNDATA_ROOT/lcmerge ; ls *.README". The minimal information needed is shown in versions ESSENCE_WV07 & SNLS_Ast06; the maximum information (for systematic studies) is shown in version SDSS_HOLTZ08. The light-curve fitter uses FLUXCAL$= 10^{(11-0.4m)}$. The scale-factor of 10^{11} is arbitrary; you can change it, but then your distance moduli will all have a common offset, although the final cosmological parameters are not affected by the choice

[11] Sample kcor-input files are in $SNDATA_ROOT/analysis/sample_input_files/kcor.

of scale-factor. Use a well-measured data point in each filter to get the FLUXCAL/FLUX ratio, and then convert FLUX → FLUXCAL for each measurement. If you try to convert magnitudes to FLUXCAL, you will have problems for small & negative fluxes.

7. Modify a sim-input and fitter-input file by substituting your survey and filters. Good luck. And don't forget to send me a post-card about your experience.

5.1 Filter Names and Rules for K-corrections

The filter names defined in the K-correction input file can be anything, but only the last character (A-Z,a-z,0-9) is propagated into the simulation and fitting program. Thus, the following filter-names are all translated into '*g*': SDSS-g, SDSSg, SDSS2.5m-g. Each rest-frame filter must have a unique last character, and each observer-frame filter must have a unique last character; however, the same last character can be used in both the rest and observer frames. For example, consider filter sets CSP-[ugri] and SDSS-[ugri]. These two sets cannot both be defined as observer-frame filters in the same K-correction file, but one set can be used for rest-frame filters that describe a light curve model, and the other set can be used for the observer-frame. The K-corrections defined after the "KCOR:" keyword define which filters are used for rest/observer-frame.

6 Cosmology Fitters

There are currently two cosmology fitters available. First is `wfit.exe`, which fits for w and Ω_M assuming a flat universe. Typing the `wfit.exe` command with no arguments lists the options. The BAO and CMB priors are currently hard-wired, but a more flexible method to specify priors may be added later. An example command is as follows,

```
wfit.exe <fitresFile> -zmin .02 -zerr .001 -snrms .15 -bao -cmb
```

which specifies using SNe with $z > 0.02$, adding a peculiar-velocity uncertainty of 300 km/s (i.e, the "`zerr`" arg is $v_{\rm pec}/c$), adding an anomalous distance-uncertainty of 0.15 mag ("`snrms`" arg), and using the BAO & CMB priors. Peculiar-velocity covariances can also be used as explained below in §6.1.

The second program, `sncosmo_mcmc.exe`, is a more general cosmology fitter based on Monte Carlo Markov chains. This fitter handles more diverse cosmologies such as time-dependent w and non-zero curvature. Sample inputs files are in

```
$SNDATA_ROOT/analysis/sample_input_files/sncosmo_mcmc/
```

Both of these cosmology fitters read self-documented "fitres" files (§7.3) that contain a redshift, distance modulus and survey index (other parameters in the fitres file are ignored).

The above cosmology fitters do not yet work on the SALT-II output since this light curve fitter does not produce a distance modulus. There are currently no `SNANA` programs that process the SALT-II fitres-output, but one can use the original "`hubblefit`" program (from Guy 2007) on the `SNANA` results if you use this namelist option in the light curve fitter:

```
SALT2_DICTFILE = 'myoutput.dictfile'
```

Recall that `hubblefit` performs a simultaneous fit for the cosmological parameters, as well as for SN properties (α, β, M). There is currently some development on new techniques for extracting cosmological results from the SALT-II light curve fits.

6.1 Peculiar Velocity Covariances

The `wfit.exe` program has an option to account for peculiar velocity correlations (`SNANA v8_10` and later). First prepare a file with the following syntax

```
COV:  SN1  SN2  MUCOVAR(12)
COV:  SN1  SN3  MUCOVAR(13)
COV:  SN1  SN4  MUCOVAR(14)
etc ...
```

where `SN#` are the SN names used in the analysis (i.e, that appear in the fitres file), and `MUCOVAR(ij)` are the covariances between the distance moduli, with units of mag^2. Only off-diagonal terms from this file are used; diagonal terms are specified from the `-zerr` and `-snrms` options. The syntax is

```
wfit.exe  <fitresFile>  -mucovar <mucovarFile>  <other options>
```

where "`mucovarFile`" is the name of the file specifying the off-diagonal covariances. The `wfit.exe` program will first check your current directory for this file; if not there `wfit` will check the public area, `$SNDATA_ROOT/models/mucovar/` . The covariances for the nearby sample have been computed by the authors in [8], and these are available in

```
$SNDATA_ROOT/models/mucovar/Hui_LOWZ_mucovar.dat
```

The minimization function is given by $\chi^2 = \sum_{ij}[\Delta_i V_{ij}^{-1}\Delta_j] - B^2/C$,[12] where $\Delta_i \equiv \mu_i^{\rm data} - \mu_i^{model}$ is the distance-modulus residual for the i'th SN, V_{ij}^{-1} is the inverse of the covariance matrix, and the term B^2/C accounts for the analytic marginalization over H_0 as discussed in Appendix A of [9]. The B and C parameters are

$$B = \sum_i (\Delta_i/\sigma_i^2) \quad \longrightarrow \quad \sum_{i,j} (\Delta_i V_{ij}^{-1}) \tag{10}$$

$$C = \sum_i 1/\sigma_i^2 \quad \longrightarrow \quad \sum_{i,j} V_{ij}^{-1} \, , \tag{11}$$

where σ_i is the diagonal-uncertainty on the distance modulus. The expressions left of the arrows (Eqs. 10-11) are from [9], while the expressions right of the arrows show the `wfit` implementation that accounts for the off-diagonal covariance terms. The B^2/C term is equivalent to re-minimizing with the weighted-average distance-modulus residual subtracted from each distance-modulus residual; $\Delta_i \rightarrow \Delta_i - <\Delta>$.

[12] We leave out the constant term $\ln(C/2\pi)$.

7 Miscellaneous Tools and Features

7.1 Command-line Overrides

The simulation and light curve fitter described in the sections above are each driven by an input file that specifies instructions and parameters. For convenience, all input-file parameters can be specified on the command line. For example, if you have

```
GENMODEL: mlcs2k2.v006
GENTAU_AV: 0.35
```

in your simulation-input file, you can override these options on the command-line without editing the input file:

```
snlc_sim.exe mysim.input  GENMODEL mlcs2k2.v007  GENTAU_AV 0.45
```

These command-line overrides are useful for quick testing, and for writing wrappers that do many analysis variations without creating a new input file for each job. An invalid or unrecognized command-line option results in an immediate abort. The command-line options are printed to stdout, and therefore if you pipe your job to a log-file, you can rerun the same job as long as you have the original input file.

7.2 K-correction Dump Utility: `kcordump.exe`

The K-correction tables are stored in `HBOOK` files, and accessing this information can be tricky even for those familiar with `PAW`. The utility `kcordump.exe` can be used to check a K-correction value for specific filter, epoch, and primary reference. If you type `kcordump.exe` with no arguments, the program will ask you for all needed arguments. You can also use command-line arguments, as illustrated here for SNLS K_{gU} at peak at $z = 0.28$:

```
> kcordump.exe HFILE_KCOR Hsiao/kcor_SNLS_Bessell90_VEGA.his  \\
               FILT_OBS g  FILT_REST U  Z .28  TREST 0.
```

The dump utility checks your current directory and `$SNDATA_ROOT/kcor` for the existence of the K-correction file (argument of `HFILE_KCOR`). All K-correction dumps are done with no spectral warping. To see K-corrections with warping, generate simulated SNe Ia and scroll through the data files for symbols containing "`KCOR`" and "`WARP`." The slight disadvantage with using the simulation to check K-corrections is that you cannot specify which K-corrections to check, but you can only compare to whatever the simulation generates.

7.3 Combining "Fitres" Files: `combine_fitres.exe`

As discussed in §4.2, the fit results are written to a self-documented "fitres" file. The simulation can also be used to dump generated variables into a file with the same format (§3.20.3). The utility `combine_fitres.exe` is useful to combine, or merge, multiple fitres files containing information about the same SNe. Note that fitres files with different SNe can be combined by simply using the unix "cat" command.

As an example, consider the following fitres files generated from different light curve fitters:

```
> more mlcs.fitres
NVAR:  2
VARNAMES: Z DLMAG
SN: 50001    0.1576  39.475
SN: 50002    0.2030  40.291

> more salt2.fitres
NVAR:  2
VARNAMES: x1 c
SN: 50001   -2.343  0.013
SN: 50002    0.893  0.235

> combine_fitres.exe  mlcs.fitres  salt2.fitres

> more combine_fitres.txt
NVAR:    5
VARNAMES: CID Z DLMAG x1 c
SN:  50001   0.157600001  39.4749985 -2.34299994  0.0130000003
SN:  50002   0.202999994  40.2910004  0.893000007  0.234999999
```

Note that although the SN candidate id (CID) is required after the "`SN:`" keyword, it is optional to specify `CID` in the `VARNAMES` list.

Fitres files with the same variable names can also be combined. The second repeated variable gets a "2" appended to the variable name, the third repeated variable gets a "3" appended, etc ... For example, consider two `MLCS2k2` fits with slightly different options,

```
> more mlcs.fitres
NVAR:  2
VARNAMES: Z DLMAG
SN: 50001    0.1576  39.475
SN: 50002    0.2030  40.291
```

```
> more mlcs_test.fitres
NVAR:  2
VARNAMES: Z DLMAG
SN: 50001    0.1576  39.829
SN: 50002    0.2030  40.443

> combine_fitres.exe  mlcs.fitres  mlcs_test.fitres

> more combine_fitres.txt
NVAR:    5
VARNAMES: CID Z DLMAG Z2 DLMAG2
SN:  50001   0.157600001  39.4749985  0.157600001  39.8289986
SN:  50002   0.202999994  40.2910004  0.202999994  40.4430008
```

Up to ten fitres files can be merged together. If there are SNe in the second (3rd, 4th...) fitres file that are not in the first fitres file, then the these SNe will be ignored. If there are SNe in the first fitres file that are not in the other fitres files, then values of -999 are stored for the missing SNe. In addition to creating a merged fitres file, a merged ntuple file (`combine_fitres.tup`) is also created. This ntuple can be analyzed with `PAW` or `root`.

7.4 Ntuple $\leftrightarrow$ Fitres Format

In addition to the fitres file output from the light curve fitter, there are also ntuples with much more extensive information about each SN. If you are familiar with `PAW`, you can immediately analyze ntuples 7788 and 7799. If you are not familiar with `PAW`, and are not in the mood to learn, you can extract interesting variables into a "fitres-formatted" text-file. The first step is to get a list of available variable using the `ntprint` utility:

```
  > ntprint.cmd  snfit.his  7788
```

Hopefully the meaning of each variable is obvious; if not, you will need to ask or look at the source code. Next, create a file containing the names of the variables that you want to extract; for example,

```
> more MYVAR.LIST
 CID SNRMAX_g  SNRMAX_r
     SNRMAX_i
```

The first variable *must* be CID, and the variable names can be separated by blank spaces and/or carriage returns. You can also use upper and/or lower case since the underlying extraction routine is case-insensitive. Finally, to extract the above variables from the ntuple into a text file,

```
> ntdump.pl  snfit.his  7788  MYVAR.LIST  HEADER
    or
> ntdump.pl  snfit.his  7788  MYVAR.LIST
```

The first option gives an output file in the self-documented fitres format; i.e, with the NVAR and VARNAMES header. If you leave out the HEADER option, then the output file has no header and no SN keywords. You can get help with both utilities by typing

```
> more  $SNANA_DIR/util/ntprint.cmd
> more  $SNANA_DIR/util/ntdump.pl
```

7.5 Appending Variables to the Standard Fitres Output

If you set the &FITINP namelist variable FITRES_DMPFILE, you will get a standard subset of variables that are adequate for a typical cosmology fitter. However, there may be more sophisticated programs that require additional information, such as covariances, error-flags, S/N ratios, etc ... Your fitres file can be appended with an arbitrary set of variables stored in the fitres ntuple (ntid 7788) or from any other ntuple. The utility script runs as follows,

```
ntappend2fitres.cmd  myappend.input
   or
ntappend2fitres.cmd  myappend.input  DUMP
```

where the input file contains the relevant input information. The DUMP option prints a list of all available variables to choose from; you may have to search to code to find the exact definition. See top of $SNANA_DIR/util/ntappend2fitres.cmd for instructions on filling out the input file.

7.6 Co-Adding SIMLIB Observations on Same Night: simlib_coadd.exe

If a survey takes many exposures per filter in one night, the resulting SIMLIB can be quite large, and there may be no benefit to simulating each exposure within a night. Since one typically combines these exposures into a single co-added exposure, there is a utility to translate a SIMLIB so that there is just one effective co-added exposure per filter per night,

```
> simlib_coadd.exe  MYSURVEY.SIMLIB
```

which produces an output SIMLIB called MYSURVEY.SIMLIB.COADD . By default, observations within 0.4 days are combined into a single co-added observation, and at least three observations are required to keep a sequence of observations (i.e, a LIBID). The CCD noise, sky-noise and zeropoint are calculated to reflect a single co-added exposure as follows,

$$\mathtt{ZPT}(\mathtt{COADD}) = 2.5 \log_{10}\left[\sum_i 10^{(0.4\cdot \mathtt{ZPT}_i)}\right] \qquad \mathtt{NOISE}(\mathtt{COADD}) = \sqrt{\sum_i \mathtt{NOISE}_i} \quad , \tag{12}$$

where i is the exposure index. The co-added PSF is simply the average of the PSF values from the individual exposures.

There are additional options to change the requirement on the minimum number of observations, to change the time-separation for co-adding, and to determine the Milky Way Galactic extinction (MWEBV) from [6],

```
> simlib_coadd.exe  MYSURVEY.SIMLIB  MWEBV  --TDIF .2  --MINOBS 5
```

When the "MWEBV" option is used, observation sequences with MWEBV>2 are rejected. Read top of $SNANA_DIR/src/simlib_coadd.c for additional options.

7.7 Bug-Catcher: the SNANA_tester Script

To help verify that the SNANA code delivers the same results with each new version, there is a testing utility, SNANA_tester.cmd (no arguments), that runs a pre-defined list of jobs with two different versions of SNANA, and reports discrepancies. Typically this script is run just before releasing a new SNANA version, but users may want to run this script on other platforms. The goal is to catch and fix unanticipated changes (i.e, bugs) before releasing each new SNANA version. The top-level instruction file is here,

```
$SNDATA_ROOT/SNANA_TESTS/SNANA_TESTS.LIST
```

which specifies a series of test-jobs, input files, and log-file parsing instructions to extract results to compare between SNANA versions. The SNANA_tester.cmd script is written for the Fermilab ups product system; on other platforms, script modifications will be needed to setup the correct versions of SNANA. Users who use a particular feature of SNANA should check if the current test-jobs provide adequate protection; if not, you may request additional test-jobs.

7.8 Data Archival: version_archive.cmd

When submitting a paper for publication, it is highly recommended to archive the SN data version(s) used in the analysis. The archive ensures that the results can be reproduced at any future time, and allows a continued analysis in the future as new ideas become available. As an example, if you write a paper about SNe Ia discovered by the SDSS-II SN survey and observed spectroscopically with KECK, the archival command is

```
version_archive.cmd  KECK-SDSS  KECK-SDSS_Paper09
```

The first argument is the current version name, and the second argument is the name of the archive that will be created in

```
$SNDATA_ROOT/lcmerge/archive/KECK-SDSS_Paper09.tar.gz
```

The KECK_SDSS version may never change, or it may change, for example, if the SN photometry is upgraded. The archive-version, however, will never change. For users who download SNDATA_ROOT, i.e., who do not work on the development server, some extra care must be taken to save your archive version(s) before downloading a new SNDATA_ROOT. The safest mechanism is copy (or send) the archive tarball to the server used for SNANA development.[13] Archiving on the development machine is recommended so that your archive-versions will be distributed (via SNDATA_ROOT) to all SNANA users.

[13] The current SNANA development server is sdssdp47.fnal.gov, but this may change in the future.

In addition to archiving the data version(s) related to a paper, you should also make an archive of your input files, scripts, log-files and hbook outputs from the fitter, simulation and any other programs used in the analysis. There is no mechanism in SNANA to make such a backup, and it is therefore up to each user to prepare such an archive.

7.9 Preparing Non-Ia Templates for the Simulation

... coming soon ...

7.10 Translating SNDATA files into SALT-II Format

The snana.exe program can convert SNANA-formatted data (or simulation) files into SALT-II format needed to run the original (Guy07) SALT-II fitter code and the SALT-II training code. A sample namelist is as follows:

```
&SNLCINP
   VERSION_PHOTOMETRY = 'SDSS_HOLTZ08'
   OPT_REFORMAT_SALT2 = 2
   REFORMAT_KEYS  = '@INSTRUMENT SLOAN  @MAGSYS AB'
   SNFIELD_LIST = '82N' , '82S'
   cutwin_cid   =  0, 100000
&END
```

Note that OPT_REFORMAT_SALT2=2 is for the newer SALT-II format with one file per SN (snfit version 2.3.0 and higher), while OPT_REFORMAT_SALT2=1 is for the original format with one file per passband.

7.11 SIMSED Spectrum Extraction: SIMSED_extractSpec.exe

For a SIMSED model, the program SIMSED_extractSpec.exe can be use to extract a single spectrum for a particular SIMSED model version, epoch, and set of parameter values corresponding to the PARNAMES in the SED.INFO file. The current program uses the parameter values on the SED.INFO grid that are closest to the user-specified parameters; a future version may interpolate for better accuracy. See usage instructions at top of $SNANA_DIR/src/SIMSED_extractSpec.c .

7.12 SIMSED Fudge Afterburner: SIMSED_fudge.exe

For a given SIMSED model (§8.4), an arbitrary color law can be applied to generate a new SIMSED version. The program syntax is

```
SIMSED_fudge.exe <inFile>
```

where an example input file is here:

```
$SNDATA_ROOT/analysis/sample_input_files/SIMSED/colorFudge.input
```

7.13 Splitting a Large Sample into Sub-versions: split_version.pl

While the SNANA simulation can generate up to 900,000 SNe, the analysis programs (snana.exe & snlc_fit.exe) can process only 3100 per job. Therefore large data versions must be analyzed with multiple jobs. The simplest way to run multiple analyses on a large data version is to specify a unique CUTWIN_CID range for each job. The disadvantage of using CID ranges is that each job must read all of the data files, and this can add significant overhead.

A more efficient method for processing multiple jobs is to split a large version into sub-versions using the following utility:

```
split_version.pl  <VERSION>   <NSPLIT>
```

where VERSION is the name of the version, and NSPLIT is the number of sub-versions to create. Note that the data files are NOT copied; each split version contains only the auxiliary files, and the list file names point to the original files. The name of each sub-version is SPLIT*nn*_[VERSION] where *nn* is the split-index that goes from 01 to NSPLIT. If the sub-versions still have more than 3100 files, you will get a warning; it does not abort because non-SNANA codes may allow for more SNe per version.

After splitting version "TEST", each split sub-version should be processed as a separate job as follows:

```
snlc_fit.exe snfit.nml  VERSION_PHOTOMETRY SPLIT01_TEST \
                        HFILE_OUT SPLIT01.his   FITRES_DMPFILE SPLIT01.fitres

snlc_fit.exe snfit.nml  VERSION_PHOTOMETRY SPLIT02_TEST \
                        HFILE_OUT SPLIT02.his   FITRES_DMPFILE SPLIT02.fitres

etc ...
```

These jobs can run in parallel, and the arguments of HFILE_OUT and FITRE_DMPFILE above can be arbitrarily modified.

7.14 Re-numbering Simulated SNIDs: simid_renumber.pl

A simulated sample can have its SNIDs re-numbered with the command

```
simid_renumber.pl  <VERSION>  <SNID_START>
```

This script modifies the simulated "VERSION" data files so that SNID starts at the indicated value, and increments by 1 for each successive file. The ${VERSION}.LIST file is modified, and the 'SNID:' argument in each data file is modified.

To illustrate the utility of this script, consider multiple mixed-SN samples, with 900,000 SNe generated for each sample. Next suppose only 9,000 (1%) per sample are written out after trigger/selection criteria. The SNIDs will be unique within each sample, but the same SNID may reside in different samples, causing potential confusion if all samples are analyzed together. To avoid repeated SNIDs, the above script can re-number those 9000 SNe to go from 1–9000, 10000–19,000, 20000–29000, etc ..., thereby ensuring a unique SNID among all samples.

As a precaution, the original sim-directory is saved as $SNDATA_ROOT/SIM/${VERSION}_SAVE. These "_SAVE" directories should be removed once you have confidence in the modified SNIDs. Note that you cannot use SNANA to analyze ${VERSION}_SAVE unless you modify the auxiliary files[14] to have the same "_SAVE" suffix.

[14]The auxiliary files are ${VERSION}.LIST, ${VERSION}.README, ${VERSION}.DUMP, and ${VERSION}.IGNORE.

8 Light Curve Models

Here we discuss some of the available light curve models for the simulation and fitter. This is only brief a technical discussion on how to use the models in SNANA; a reference for each model is provided for more details. For each model "MMM" there is a dedicated model directory,

```
$SNDATA_ROOT/models/MMM
```

containing one or more versions of the model. The file MMM.default points to a default version if a generic model name (e.g.,SALT2, mlcs2k2, snoopy) is specified. Any model version can also be selected. Examples of model selection are

```
GENMODEL:       MMM      # sim-input; use what's in MMM.default
GENMODEL:       MMM.v01  # sim-input; use this version

FITMODEL_NAME = 'MMM'      ! fit-input; use what's in MMM.default
FITMODEL_NAME = 'MMM.v02'  ! fit-input; use this version
```

In the sub-sections below, "xxx" refers to a floating point number that must be specified by the user. In general, each simulated parameter "XXX" is specified by three input keys: 1) GENMEAN_XXX, 2) GENRANGE_XXX, and 3) GENSIGMA_XXX. The GENSIGMA key has two values to specify an asymmetric (or symmetric) Gaussian distribution. The GENRANGE values truncate the distribution.

8.1 MLCS2k2

Reference: Jha, Riess, Kirshner, **AJ 659**, 122 (2007).
Simulation input keys for `snlc_sim.exe` :

```
GENMEAN_DELTA:    xxx         # shape/luminosity parameter
GENRANGE_DELTA:   xxx  xxx
GENSIGMA_DELTA:   xxx  xxx

GENMEAN_RV:       xxx        # CCM89 dust parameter
GENRANGE_RV:      xxx  xxx
GENSIGMA_RV:      xxx  xxx

GENRANGE_AV:      xxx  xxx     # CCM89 V-band extinction
GENTAU_AV:        xxx          # dN/dAV = exp(-AV/xxx)
```

&FITINP namelist variables for `snlc_fit.exe` :

```
OPT_SNXT       = 1   ! Use CCM89 + ODonnel94 update
SCALE_COVAR    = 4.1 ! scale cov matrix
OPT_LANDOLT    = 1   ! 1=>transform Bessell90 <=> Landolt with color transf.
                     ! 3=> same for mlcs model & nearby-Landolt mags

OPT_PRIOR_AV   = 1   ! 0=> switch off AV prior
NGRID_PDF      = 11  ! marginalize NGRID per variable (0 => fit min only)
NSIGMA_PDF     = 4   ! initial guess at integration range: +- 4 sigma
OPT_SIMEFF     = 1   ! use simulated eff as part of prior
PRIOR_AVEXP  = 0.3   ! tau of exponential AV prior
PRIOR_AVRES  = 0.005 ! smooth Gauss rolloff for AV<0.
PRIOR_MJDSIG = 10.   ! Gauss prior on MJD at peak

INISTP_RV      = xxx  ! 0 => fix RV to INIVAL_RV
INIVAL_RV      = 2.2  !
INISTP_AV      = xxx  ! 0 => fix AV = INIVAL_AV in fit; else float
INIVAL_AV      = xxx
INISTP_LUMIPAR = xxx  ! 0 => fix DELTA to INIVAL_LUMIPAR; else float
INIVAL_LUMIPAR = xxx

PRIOR_DELTA_PROFILE = xxx, xxx, xxx, xxx  ! grep snlc_fit.car for details
```

8.2 SALT-II

Reference: J. Guy et al., **A&A 466**, 11 (2007).
Simulation input keys for `snlc_sim.exe` :

```
GENMEAN_SALT2x1:   xxx         ! stretch parameter
GENRANGE_SALT2x1:  xxx  xxx
GENSIGMA_SALT2x1:  xxx  xxx

GENMEAN_SALT2c:    xxx         ! color parameter
GENRANGE_SALT2c:   xxx  xxx
GENSIGMA_SALT2c:   xxx  xxx

# mag = mB* + alpha*x1 - beta*color
GENALPHA_SALT2:   xxx   ! x1 coeff
GENBETA_SALT2:    xxx   ! color coeff
```

&FITINP namelist variables for `snlc_fit.exe` :

```
SALT2alpha     = xxx  ! used only to compute mu - muref
SALT2beta      = xxx  ! idem

INISTP_COLOR   = xxx  ! 0 => fix color to INIVAL_COLOR; else float
INIVAL_COLOR   = xxx
INISTP_LUMIPAR = xxx  ! 0 => fix x1 to INIVAL_LUMIPAR; else float
INIVAL_LUMIPAR = xxx

PRIOR_MJDSIG   = xxx            ! Gaussian prior on MJD at peak
PRIOR_LUMIPAR_RANGE = xxx, xxx  ! flat prior on x1 (prevents crazy values)
PRIOR_LUMIPAR_SIGMA = xxx       ! Gauss rolloff at edges of flat prior

SALT2_DICTFILE = 'xyz'  ! output formatted for original hubblefit
```

Note that the `INISTP_XXX` and `INIVAL_XXX` parameters are specified only to fix these parameters in the fit. If not specified, these parameters are floated in the fit.

8.3 SNooPy

Reference: C. Burns et al., arXiv:1010.4040.
Simulation input keys for `snlc_sim.exe` :

```
GENMEAN_DM15:    xxx         # shape/luminosity parameter
GENRANGE_DM15:   xxx  xxx
GENSIGMA_DM15:   xxx  xxx
GENMEAN_RV:       xxx        # CCM89 dust parameter
GENRANGE_RV:      xxx  xxx
GENSIGMA_RV:      xxx  xxx
GENRANGE_AV:      xxx  xxx     # CCM89 V-band extinction
GENTAU_AV:        xxx          # dN/dAV = exp(-AV/xxx)
```

&FITINP namelist variables for `snlc_fit.exe` :

```
OPT_PRIOR_AV   = 1   ! 0=> switch off AV prior
NGRID_PDF      = 11  ! marginalize NGRID per variable (0 => fit min only)
NSIGMA_PDF     = 4   ! initial guess at integration range: +- 4 sigma
OPT_SIMEFF     = 1   ! use simulated eff as part of prior
PRIOR_AVEXP  = 0.3   ! tau of exponential AV prior
PRIOR_AVRES  = 0.005 ! smooth Gauss rolloff for AV<0.
PRIOR_MJDSIG = 10.   ! Gauss prior on MJD at peak
PRIOR_LUMIPAR_RANGE = 0.7, 2.0 ! constrain DM15 in this range
PRIOR_LUMIPAR_SIGMA = 0.7, 2.0 ! Guass roll-off sigma for DM15 prior
INISTP_RV      = xxx  ! 0 => fix RV to INIVAL_RV
INIVAL_RV      = 2.2  !
INISTP_AV      = xxx  ! 0 => fix AV = INIVAL_AV in fit; else float
INIVAL_AV      = xxx
INISTP_LUMIPAR = xxx  ! 0 => fix DELTA to INIVAL_LUMIPAR; else float
INIVAL_LUMIPAR = xxx
```

Without a tuned ATLAS library (for gsl), the SNooPy generator is very slow such that a single light curve fits takes several minutes. To speed up the generator, the SNooPy model can be parameterized on a three-dimensional grid (filter, epoch, ΔM_{15}) using the GRID option from §3.24. This grid is specified by the GRIDFILE keyword in the SNooPY.INFO file that resides in the same directory as the model templates; both the simulation and fitter interpolate the GRID for each set of parameters. Also note that SNooPy returns a relative flux normalized to one at peak; the conversion to absolute magnitude is given for each filter in the SNooPY.INFO file.

8.4 SIMSED

A SIMSED model contains a full sequence of spectra for each epoch. These models typically result from specialized explosion-model codes such as FLASH, Sedona and Pheonix. Currently the SIMSED model is used only in the SNANA simulation; implementation in the fitter may come later when these models are more reliable. Each SIMSED version resides in

```
$SNDATA_ROOT/models/SIMSED
```

and contains a sequence of SEDs corresponding to parameters that describe the explosion model. The parameters are quantities such as Ni-56 mass, viewing angle, kinetic energy, and extinction. The only limit to the number of parameters (and SEDs) is the amount of memory on your computer.

Each SIMSED version contains an SED.INFO file specifying the list of parameter names, and the parameter values for each SED. An example of an SED.INFO file is as follows:

```
NPAR: 5
PARNAMES: MNI COSANGLE MBSED DM15SED TEXPL

SED: GCD2D_Ni0.47_Pre80_1.SED  0.47  -0.967  -18.350  0.571  -21.628
SED: GCD2D_Ni0.47_Pre80_8.SED  0.47  -0.500  -18.413  0.574  -20.977
etc ...
```

Users must prepare the SED.INFO file and the SEDs,[15] as there is no standard SNANA code for this task.

To simulate a SIMSED model, the distribution for each parameter must be specified in the sim-input file as follows,

```
SIMSED_PARAM:  MNI         # mass of Ni56
GENMEAN_MNI:  0.9          # mean of bifurcated Gaussian
GENRANGE_MNI: 0.47 1.26    # generation range
GENSIGMA_MNI: 0.4  0.25    # bifurcated Gaussian

SIMSED_GRIDONLY:  COSANGLE        # cosine of viewing angle
GENMEAN_COSANGLE:   0
GENRANGE_COSANGLE: -0.96  0.96
GENSIGMA_COSANGLE:  1.E7  1.E7   # large sigmas => flat distribution

# baggage parameters
SIMSED_param:  MBSED     # calculated peak mB
SIMSED_param:  DM15SED   # calculated DM15
SIMSED_param:  TEXPL     # time of explosion relative to peak
```

[15] The SED format is : epoch(days) wavelength(Å) flux(erg/cm^2/s/Å).

If you do not specify each parameter, or you make a mistake typing the parameter name, the simulation will abort. For the SIMSED_PARAM keyword, the simulation will interpolate magnitudes as a function of the explosion-model parameter, resulting in a continuous distribution of the specified bifurcated Gaussian. For the SIMSED_GRIDONLY keyword, only values at the grid nodes are generated. This GRIDONLY option may be useful in cases where an integer flag specifies a random ignition point, and therefore continuous interpolation makes no sense. Also note that for the GRIDONLY option, the specified bi-Gaussian distribution is respected; each randomly chosen parameter is 'snapped' to the nearest grid value.

The SIMSED_param keyword specifies a "baggage" parameter. These parameters are ignored in the generation, but the interpolated values are computed and stored along with the other parameters. These variables appear in the fitres ntuple, and can be included in the SIMGEN_DUMP list (§3.20.3)

There are two internal binary files to speed up the initialization. In principle this all works behind the scenes, but it is explained here in case there are problems. The initialization takes about 1 second per SED, and will be rather annoying if/when there are 100's or 1000's of SEDs to initialize. About half the time is reading the SED text files, and the other half is spent doing all the flux-integrals as a function of passband, redshift, Trest, and SED surface. The first time that a new SIMSED version is simulated, the initialization will be slow, but two binary files are created to speed up future runs. The first binary file contains the SEDs (SED.BINARY), and it is stored in the same version (VVV) directory, as the SED text files, $SNDATA_ROOT/models/SIMSED/VVV. This SED.BINARY file will be automatically used by any SNANA user who specifies the VVV version.

The second binary file is the flux-integral table, and this file is stored in YOUR local directory. The reason for storing in your directory is that it depends on the survey and filters, and therefore this file is specific to your analysis. The validity of each binary file is verified internally; if there is an inconsistency between the two binary files and/or the requested survey parameters, the simulation will abort and recommend removing the old binary file(s) so that new ones can be created. Beware that for SIMSED models with many SEDs, the locally created binary file may exceed your home disk quota; in this case, run the first simulation from a scratch disk and then define a symbolic link from your home directory to the binary file on the scratch disk.

If you are having problems with the binary files and just want to use the text files, the use of binary files can be switched off with

```
snlc_sim.exe  mysim.input  SIMSED_USE_BINARY 0
```

9 SNANA Updates

The SNANA software is released in incremental versions, reflecting improvements, new code, and bug fixes. With each new version there is an e-mail reminder sent to the SNANA@FNAL.GOV mailing list. Users are encouraged to sign up for the SNANA mailing list by asking any co-author of the overview paper (arXiv:xxxxx). The SNANA mailing list can also be used to ask for help, report bugs, suggest changes, etc ... Details of each release can be found by doing

```
tail -100 $SNANA_DIR/doc/README_UPDATES
```

Users should develop download & setup scripts to easily maintain the most current version of SNANA so that you don't get trapped with an old or obsolete version. The SNDATA_ROOT tarball is updated less frequently; README_UPDATES will indicate if and why a new SNDATA_ROOT is needed.

This manual ($SNANA_DIR/doc/snana_manual.pdf) is updated mostly in response to questions from users. If you spot mistakes or obsolete information in the manual, please report it immediately !

10 Reporting Problems

Please don't hesitate to report software problems or obsolete/incorrect information in the manual. If the problem is related to an abort or crash, please include a tarball that contains the input file(s) and data file(s) that reproduce the problem, as well as a log-file with the program's screen-dump. Indicate which SNANA version was used, and try to isolate the problem in a single data file to avoid sending large numbers of data files. **WARNING: if you don't provide enough information to reproduce the problem, we will not be able to provide assistance.**

If the problem is related to processing simulated data files, please do the following. First, manually create a special version called SIM_DEBUG that resides in SNDATA_ROOT/SIM/SIM_DEBUG. Copy the relevant data file(s) that cause the problem, such as unexpected abort or crazy fitted values. In the same directory, manually create the needed auxiliary files as follows:

```
touch  SIM_DEBUG.README
touch  SIM_DEBUG.IGNORE
ls *DAT >! SIM_DEBUG.LIST
```

Finally, send this directory, along with the needed input files, to one of the developers.

References

[1] S. Jha, A. G. Riess, and R. P. Kirshner. Improved Distances to Type Ia Supernovae with Multicolor Light Curve Shapes: MLCS2k2. *AJ*, 659:122, 2007.

[2] J. Guy et al. SALT2: Using Distant Supernovae to Improve the use of Type Ia Supernovae as Distance Indicators. *A&A*, 466:11–21, 2007.

[3] J. Prieto et al. A New Method to Calibrate the Magnitudes of Type Ia Supernovae at Maximum Light. *AJ*, 647:501–512, 2006.

[4] G. Goldhaber et al. Timescale Stretch Parameterization of Type Ia Supernova B-band Light Curves. *Astrophys. J.*, 558:359–368, 2001.

[5] B. Hayden, P. Garnavich, and SDSS-SN Survey. The Rise and Fall of SDSS-II Supernovae. *Bulletin of the American Astronomical Society*, 41:254, 2009.

[6] D. J. Schlegel, D. P. Finkbeiner, and M. Davis. Maps of Dust Infrared Emission for Use in Estimation of Reddening and Cosmic Microwave Background Radiation Foregrounds. *Astrophys. J.*, 500:525, June 1998.

[7] P. Nugent et al. K-Corrections and Extinction Corrections for Type Ia Supernovae. *PASP*, 114:803, 2002.

[8] L. Hui and P. B. Greene. Correlated Fluctuations in Luminosity Distance and the Importance of Peculiar Motion in Supernova Surveys. *Phys. Rev.*, 73(12):123526, 2006.

[9] M. Goliath et al. Supernovae and the Nature of the Dark Energy. *A&A*, 380:6–18, 2001.